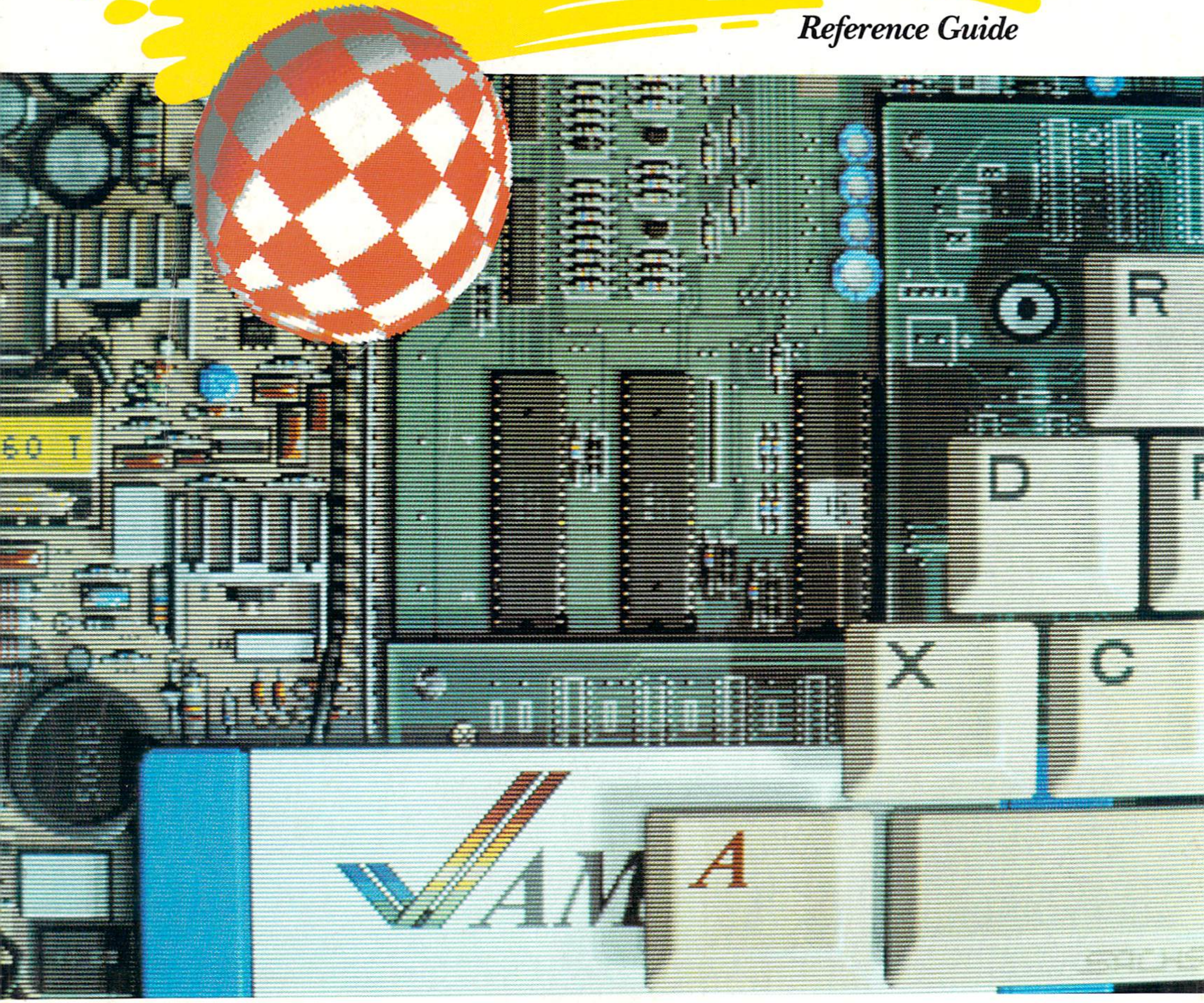
— SPECIAL ISSUE —

1987
U.S.A. $3.95
A CWC/I
Canada $4.50
Publication

AMIGA WORLD

Reference Guide



FREE! Pull-Out Reference Guide!

"All the tools you need to print"

New Laser Time

Scribble!, TextCraft, Deluxe Paint, Aegis Images, PageSetter, and LaserScript are registered trademarks of Micro-Systems Software Inc., Commodore/Amiga Inc., Electronic Arts, Aegis Development, and Gold Disk Inc. respectively.

VOL. CXXVII No. 1

LASER BARRIER BROKEN

Historic first: Amiga creates typeset quality text with laser printer

You are reading history. You are reading the first published examples of typeset quality, laser printed words created on a desktop publishing program for the Commodore Amiga. Yesterday, it was impossible. Today, it's within every Amiga owner's reach with Gold Disk's PageSetter 1.0 Desktop Publishing Software and the breakthrough new PageSetter LaserScript program. Reports, forms, flyers, and even newspapers can be created with a whole new level of professionalism.

PageSetter LaserScript adds an exciting new dimension to the already powerful PageSetter 1.0. In addition to providing the four most popular laser typefaces (with more on the way), PageSetter LaserScript gives you complete control over type style and size. It also frees you to change the position and size of entire pages, including the power to rotate them if you want. There's even an option to create custom templates, a time-saving feature the frees you from repetitious formatting and re-formatting. And it works with any PostScript laser printer.

"PageSetter LaserScript is just the beginning of a series of breakthrough desktop publishing programs from Gold Disk", reported a company spokesperson. "We are committed to making the Amiga the premier desktop publishing machine, and we have products in development that are going to do just that." A reliable inside source reports that a big announcement is due in the near future.

Prices are extremely competitive. PageSetter 1.0 remains priced at $149.95, while the new PageSetter LaserScript will be priced at $44.95.

In related news, Gold Disk also announced the availability of a special disk containing over 25 fonts for dot-matrix printers including the Times, Helvetica, and Courier typefaces. Gold Disk is calling their new product FontSet I and is pricing it at $34.95.

EMERGENCY TELEPHONE NUMBERS SET UP TO COPE WITH DEMAND

Gold Disk Inc has set up emergency telephone numbers to help anxious Amiga owners locate their nearest PageSetter 1.0 retailer.

"Anyone in the continental United States can reach us toll-free 1-800-387-8192", said a Gold Disk spokesperson. "All other callers should dial 1-416-828-0913" Customers can also reach Gold Disk by addressing mail to: Gold Disk Inc, P.O. Box 789, Streetsville Ontario, Canada, L5M 2C2

PageSetter 1.0 and new LaserScript make hist

PageSetter 1.0 The Only Choice For Serious Desktop Publishing

No other program comes close

Amiga owners don't need a laser printer to take advantage of the power and simplicity of PageSetter 1.0. All that's needed is an Amiga compatible dot-matrix printer. A truly intuitive user interface makes page design a snap. In the normal mode, what you see on the screen is exactly what you get when you print out the page. You can also pull back to view the entire page, or zoom in to edit up close. There are even a variety of borders, shadows, and shading patterns to choose from. PageSetter 1.0 is a fully integrated page layout program. You can create your documents with the built-in word processor, or create graphics in the built-in graphics editor. Of course you can always import documents and drawings from other Amig programs like TextCraft, Scribble! Deluxe Paint or Aegis Images.

Shipping Now

ACQUISITION

THE ULTIMATE DATABASE FOR THE AMIGA™

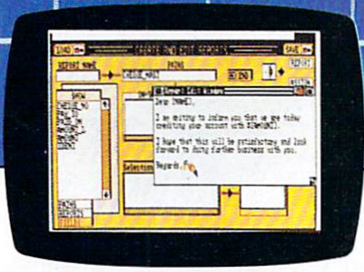
SPECIFICATION

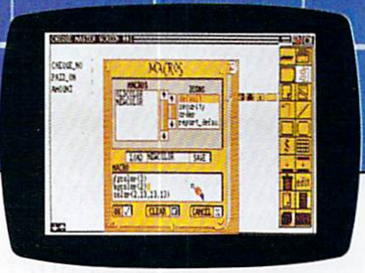
Maximum field size... 10 Megabytes
Maximum no. of fields to a record...10,000,000
Maximum no.. of records to a file... 100,000,000
Maximum size of a file... 1 Gigabyte
Maximum level of sorts... 65,000
Maximum level of selection criteria... 65,000
Maximum number of files in a system... unlimited
Maximum no. of paths attached to one file... 16
Data types: alpha/numeric, date, time, logical
Field formats: standard IFF picture, customised
Database language functions... over 200

System requirements: Amiga with at least 512K RAM and 2 floppy disk drives or a hard disk.

FEATURES

- TRULY RELATIONAL
  Multiple databases can be opened and linked together simultaneously on screen
- STORE DOCUMENTS PICTURES DATES TIMES AND CALCULATIONS.
- NEARLY 200 COMMANDS AND FUNCTIONS IN Acom
  has its own programming language for the ultimate in flexibility
- CUT AND PASTE WITHIN FIELDS PROMPTS
  manipulate data within a field and transfer to other fields, records or databases using the powerful cut and paste feature
- REPORT GENERATOR
  full function report generator allows selection across the entire filing system uses built-in text processor or any word processor
- GRAPHIC BACKDROP
  load in graphic images to act as a base screen to hold your data fields - very user friendly.





Distributed in Australia by:
COMMODORE AUSTRALIA
67 Mars Road, Lane Cove
New South Wales 2066, Australia
Distributed in the USA by:
Haitex Resources
208 Carrollton Park • Suite 1207
Carrollton, Texas 75006
(214) 241-8030

HAITEX

Distributed in Sweden:
FÖRLAGSGRUPPEN Nygatan 85
602 - 34 - NORRKÖPING
TEL: 011 13 4080

1.2 KICKSTART 512K & 2 DRIVES

* AMIGA is a trade mark of Commodore Machines

Circle 302 on Reader Service card.

Produced by
TAURUS
IMPEX LTD.
Taurus House, 3 Bridge Street, Guildford
Surrey GU1 4RY. Telephone: Guildford (0483) 579399.
Telex: 859500. Facsimile No: (0483) 502048.

Unbeatable SCSI Flexibility: No other Amiga hard drive can offer you: Capacities from 20MB to 760MB...plug-in compatibility with optical (WORM) drives, removable cartridge drives & CD ROMs...optional networking capability...dual drive compatibility with ST506 and add-ons, allowing use of up to 14 hard drives with your Amiga!

A REAL Track Record: C Ltd has been shipping Amiga hard drives since November, 1986. With thousands of units in use, you can count on C Ltd's proven hardware & software reliability.

Extraordinary Support: Call our technical support line, and you talk to the people who actually build the C Ltd products. Each drive is supplied with a complete technical manual. Each drive is *fully formatted*, with 10 MB of useful public domain software and commercial demo programs.

Cheaper By The Megabyte...

The **more** megabytes you buy, the **less** each magabyte costs you!

22MB	$ 999.95
33MB	1249.95
44MB	1499.95
50MB	1599.95
60MB	1999.95
80MB	2499.95
150MB	3299.95

additional sizes up to 750MB available upon request.

The Industry's Longest Warranty

One-year parts & labor warranty.

723 East Skinner / Wichita, Kansas 67211 / (316) 267-3807

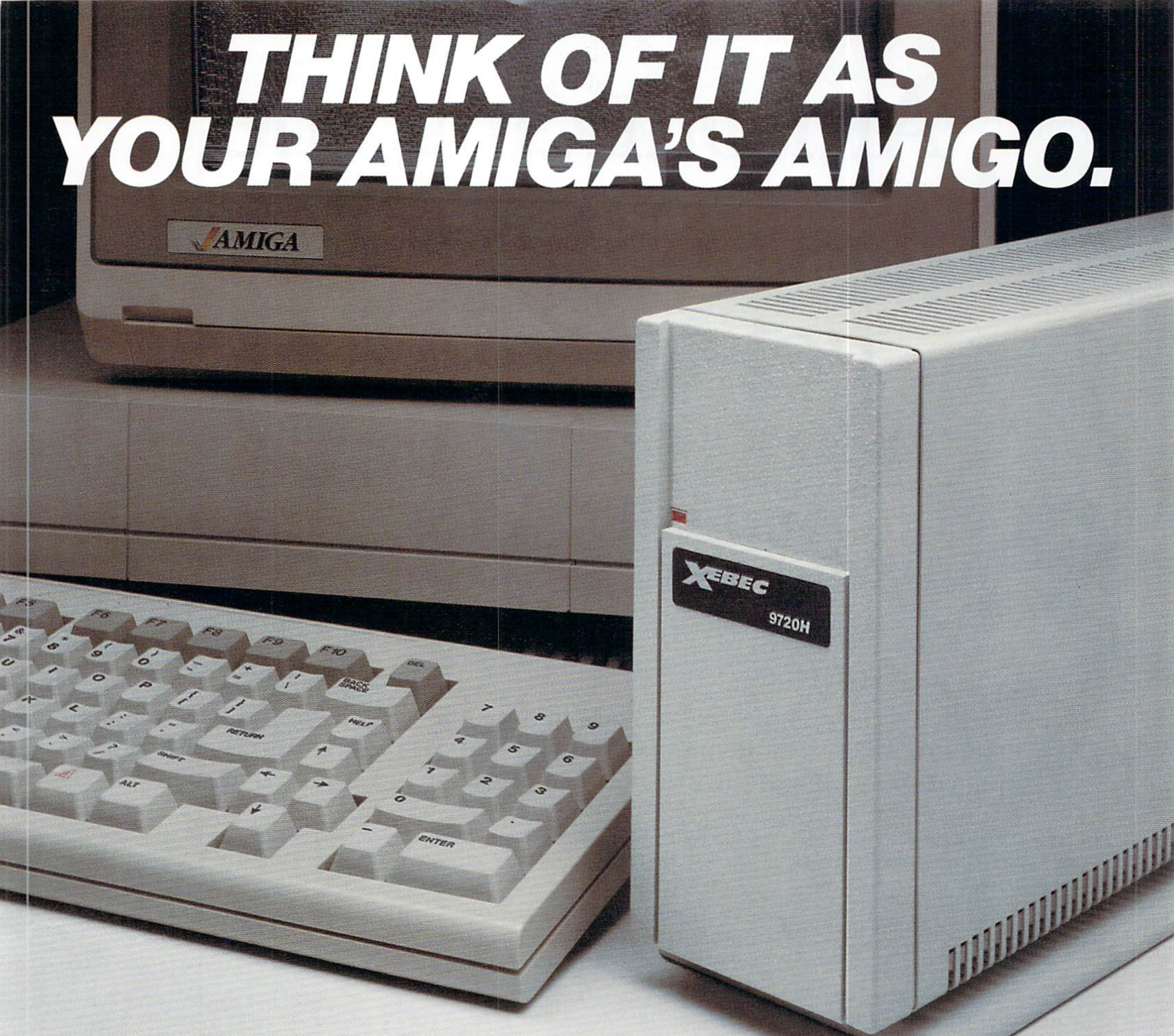
Circle 301 on Reader Service card.

SPECIAL ISSUE 1987

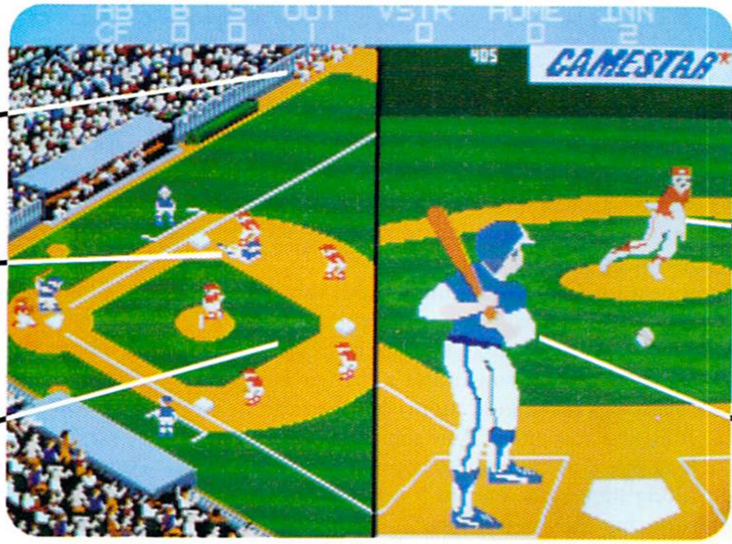
C O N T E N T S

COVER ILLUSTRATION BY JIM SACHS

AmigaWorld (ISSN 0883-2390) is an independent journal not connected with Commodore Business Machines, Inc. *AmigaWorld* is published bimonthly by CW Communications/Peterborough, Inc., 80 Elm St., Peterborough, NH 03458. U.S. subscription rate is $19.97, one year. Canada and Mexico $22.97, one year, U.S. funds drawn on U.S. bank only. Foreign Surface $39.97. Foreign Air Mail $74.97, U.S. funds drawn on U.S. bank. Phone: 603-924-9471. Entire contents copyright 1986 by CW Communications/Peterborough, Inc. No part of this publication may be printed or otherwise reproduced without written permission from the publisher. **Postmaster**: Send address changes to *AmigaWorld*, Subscription Services, PO Box 954, Farmingdale, NY 11735. Nationally distributed by International Circulation Distributors. *AmigaWorld* makes every effort to assure the accuracy of articles, listings and circuits published in the magazine. *AmigaWorld* assumes no responsibility for damages due to errors or omissions.


THINK OF IT AS
YOUR AMIGA'S AMIGO.
AMIGA
XEBEC
9720H

You Told Us What The Best Computer Baseball Game Would Have



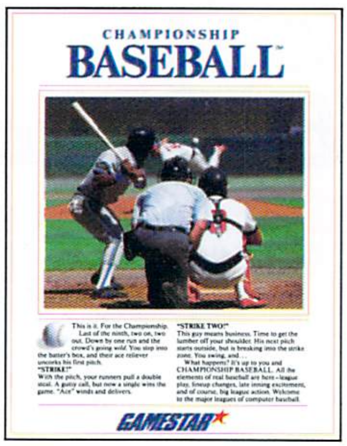
Here It Is.

CHAMPIONSHIP BASEBALL™

With 4 Divisions, 24 Teams, Playoffs and a Championship Series

You've been asking for the best in computer baseball games, and you want it *all*—graphics *and* features. We've heard you, and **Gamestar's** *Championship Baseball* delivers. No other product gives you the complete sport of baseball like *Championship Baseball:*

- Player Draft, complete with reserves.
- Setting your starting lineup.
- Batting Practice, Exhibition Game and League Play options.
- Each division is tougher than the last, so there's always a new challenge.

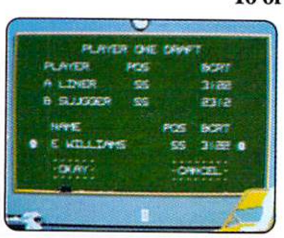


- You control all the action on the field—pitching (8 different pitches), hitting, catching, outfield relays, base running, even sliding under tags.
- Player substitutions can be made throughout the game.
- Division, playoff and championship series play.

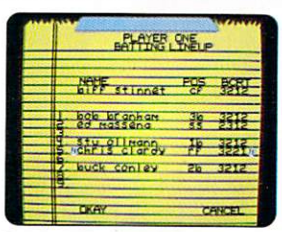
All this and much more in the most complete baseball program available. When you're ready to step into the major leagues of computer baseball, Championship Baseball's your ticket. Nobody else gives you more.

For IBM PC, Tandy 1000, Apple IIe, IIc, Commodore 64/128, Amiga, Atari ST and 100% compatible computers.

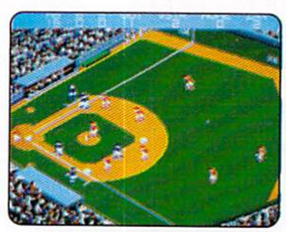
To order, visit your local retailer or call 1-800-227-6900



Select your players from the talent pool, based on their characteristics and skill ratings. Name them after today's stars, or anyone you want.



Determine the batting order that will have the best chance against your opponent.



Batting window disappears when the ball's in play, so you can see continuous action.

GAMESTAR

Circle 314 on Reader Service card.

What's So Special?

By Bob Ryan

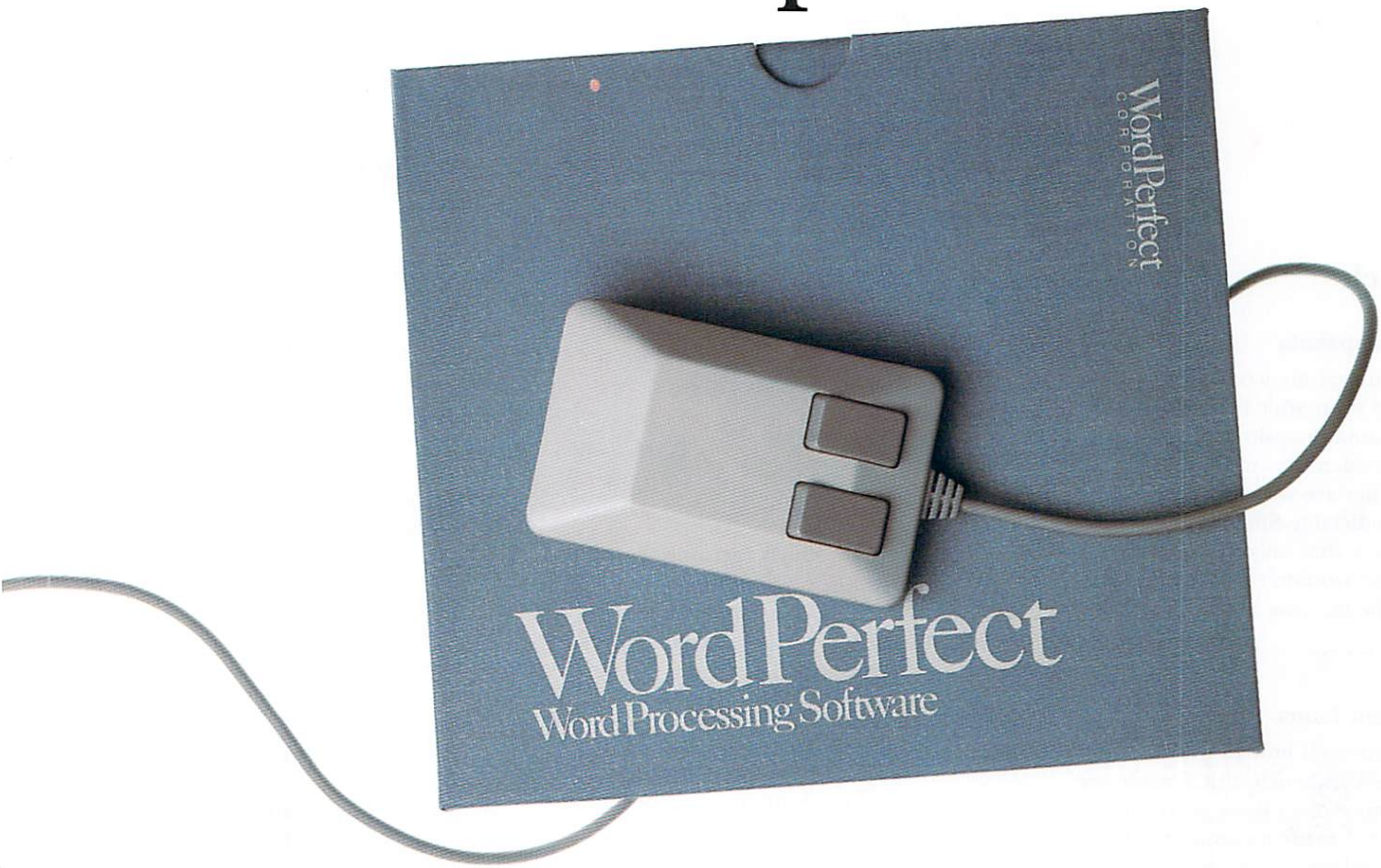
No, Guy hasn't been kidnapped by a band of crazed typesetters, nor has he absconded to Rio with last month's subscription receipts. He simply thought it would be a good idea if the Technical Editor wrote the editorial for this Special Issue of *AmigaWorld.*

You'll notice a lot of unusual things about this edition of *AmigaWorld*. Much of the normal editorial content is missing—no new product listings, no reviews, not much of anything that constitutes a regular issue. We intentionally left those (and more) out, because we wanted to limit the amount of time-sensitive material in this issue; we hope you'll keep this copy by your computer for a long time, and refer to it often.

While producing the Special Issue, we came to think of it as "a beginner's guide to everything." We chose the articles published here because they're all good introductions to different aspects of the Amiga system. Depending upon your level of expertise, you may enjoy "Beginner's Guide to the CLI," "BASIC Intuition," or perhaps "Executive Control: Introduction to the Amiga's Kernel." We think everyone will like the pull-out Technical Reference Guide bound into the issue; it brings useful information from many different sources together in one handy package. (Some of the staff didn't find the listings of hardware locations terribly useful, but I think they're neat.) The only holdover from our regular issue is Tips and Techniques: all the best from *AmigaWorld*'s Hors d'oeuvres column. In addition to reprinting and reorganizing tips from past issues, we've included a lot of new material as well.

The *AmigaWorld Special Issue* is a beginner's guide in another sense, too: It's the first time I've had executive responsibility for a magazine. I had a lot of fun bugging the other editors about deadlines (that's a switch) and working closely with the talented people in the *AmigaWorld* art department. If enough people enjoy this Special Issue (as I hope you will), the folks upstairs might just let us do it again next year.■

The start of a perfect friendship.



WordPerfect for the Amiga®

If you've been looking for a *truly* full-featured word processor that maximizes the capabilities of your Amiga, then meet WordPerfect. The same power and features that have made WordPerfect the best seller for IBM PCs are now available in a specially designed version for the Amiga computer.

A True Professional.

Regardless of your needs, WordPerfect offers you professional features to make your job easier and increase your productivity.

In the office, WordPerfect lets you automate time-consuming tasks, streamline difficult work, and maintain even the largest documents. At school, you can focus on what you're saying while WordPerfect handles the formatting. Even for uncomplicated tasks at home, WordPerfect produces professional quality results with simplicity and elegance.

WordPerfect bends to the most creative imaginations and most demanding tasks by offering such useful features as: Automatic Footnotes/Endnotes • Table of Contents • Indexing • Paragraph/Outline Numbering • Macros • Mail-Merge • Math Columns (with Four-Function Math) • File Management • Thesaurus • 115,000-word Speller • Text Columns.

A Perfect Match.

WordPerfect for the Amiga fully supports the Amiga interface, including pull-down menus and mouse control. Both the mouse and the function keys give you complete access to all WordPerfect features. And, as you'd expect, WordPerfect takes advantage of the Amiga's multi-tasking capabilities.

In addition, the Amiga version of WordPerfect offers file compatibility with WordPerfect 4.1 for the IBM.

Friends For Good.

So find out what *full-featured* really means, and get acquainted with WordPerfect. You'll be amazed to see how friendly a professional word processor can be. For more information, call or write WordPerfect Corporation, 288 West Center St., Orem, Utah 84057 (801) 225-5000.

WordPerfect
CORPORATION

AMIGA is a trademark of Commodore-Amiga, Inc.

Circle 308 on Reader Service card.

Amiga Tips And Techniques

Ideas, notions, advice and serendipitous revelations from the clever readers of AmigaWorld, *as seen in our tasty* **Hors d'oeuvres** *column. Here they are, toothpicks and all.*

Workbench

Scrambled Snapshots

I have found that my icons are stacked and scrambled even after I have cleaned them up and used Snapshot. I finally tracked the problem down; it appears that if any of the files are write-protected, you may get unpredictable Snapshot results. You would think that only the write-protected file icons wouldn't move, but this seems not to be the case.

Robert L. Pleatman
Cincinnati, OH

Saving Custom Icons

When I discovered how to use the Icon Editor on the Workbench disk, I made useful icons for my Amiga Basic programs. However, when I made an adjustment in the program and resaved it, my custom icon was replaced with the standard flow-chart icon that is created for every Amiga Basic program when it is initially saved.

There are two solutions to this problem. One is to keep an icon library and use the Icon Editor to replace the standard icon with your custom icon.

The other solution is implemented without leaving Amiga Basic, and leaves you with an old (unmodified) copy of the program.

First, get into the intermediate mode and type SAVE OLD. Now you have a new copy of the program with a standard icon and an old copy of the program with your custom icon. All you have to do now is switch the programs.

In the intermediate mode, type:

NAME OLD AS TEMPORARY
NAME *your program's name* AS OLD
NAME TEMPORARY AS *your program's name*

Now you have the new program with the custom icon and the old program with the standard icon, so you can move the old copy somewhere else or into the trash until you know your modifications work correctly.

Derek Buckley
Spokane, WA

Icon Assortment

Like many, I am slowly learning the nuances of the Amiga's operating system—the hard way. Handling Icons seems to be shrouded in mystery. Here are some hints:

1) When copying files from the CLI use:

COPY *filename*#? TO *destination*

The #? wildcard characters insure that the *filename*.info (which contains the icon information) is also copied.

2) If you create a directory from the CLI, an icon is not automatically generated for you. You can use the COPY command to "clone" an icon using:

COPY *drawer*.info TO *destination directory/ filename*.info

This makes a copy and changes the name at the same time.

3) If you have project files (like Textcraft documents, Notepad notes, BASIC programs, etc.) without icons, you can create icons for them by loading them into their tools (programs) and then resaving them. To load a program with a file from the CLI, type the program name followed by a space, and then the filename, e.g., UTILITIES/NOTEPAD *filename*, or TEXTCRAFT *filename*.

4) Programs (tools) without icons represent more of a challenge. A couple of excellent public-domain utilities are available to do this (such as IconExec) that will create bootable icons for most programs and DOS batch files.

5) One last CLI note. I like to be able to identify non-standard C directory commands that I have collected by capitalizing their names, but the AmigaDOS RENAME command won't work to just change case. The simple solution is to rename the command as something else, then rename it back to what you want in the case you want.

Frank Turner
Santa Cruz, CA

Icon System Solution

If you put Amiga Basic in a drawer, the Amiga won't find it when you try to open it by clicking on a BASIC program icon, unless you change Amiga Basic's default location. Here's how:

With Workbench loaded, first click once on the icon for the BASIC program. Next, select Info from the Workbench menu by highlighting it and releasing the mouse button. When the Info window comes onto the screen, change the default tool from :AMIGABASIC to *:your drawer name*/AMIGABASIC. This will direct the Workbench to load Amiga Basic from the drawer. I tried it and it works for me.

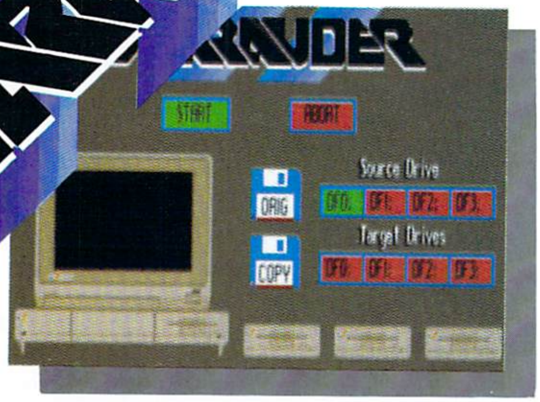
Dr. Michael J. Doyle
Bel Air, MD

RAM Disk Icons 1.2

One of the many nice additions in the version 1.2 release of AmigaDOS is an icon for the RAM disk. The icon is generated when you first access the RAM: device from the CLI. This icon does not automatically appear when you boot on a normal Workbench disk, but you can make it do so by adding the line DIR RAM: to the startup-sequence.

The RAM: device is empty when you boot, so this command does nothing but ►

The Mirror Copier Can Now Back Up A Disk Almost As Fast As Marauder II, And It Only Costs About 25% More!



Marauder II is the most powerful copier ever produced for Amiga. It will automatically copy ALL software released to date , and it requires no hardware modification of any kind. It produces completely unprotected copies of most Amiga software faster and better than any other copier.

No other copier can copy as much software as Marauder II.

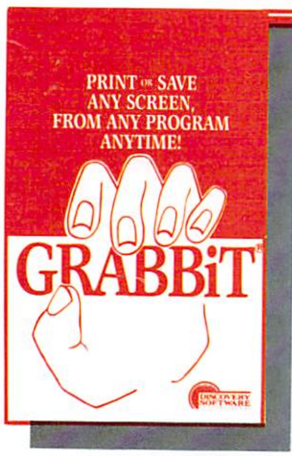
Marauder II also has the most advanced user interface your money can buy. If you have an Amiga you already know how to use Marauder II. You never have to reboot your machine to use Marauder II, it is completely compatible with the Amiga's multitasking operating system.

Marauder II has been designed with your future needs in mind. As protection schemes change you can update the program yourself with our unique "Strategy Files." The Strategy Files are developed as new software is released so that you can get them quickly and easily when you need them.

Compare the features of Marauder II to our competition and you'll see that Marauder II is quite simply the best copier you can get, at any price! And for only **$39.95** you can rest assured that your software investment is safely protected against damage, loss or theft. Don't wait, order now!

SEE FOR YOURSELF

	Marauder II	Mirror
Duplication Speed	83 Sec.	1 Minute 48 Sec.
Upgradable With Strategy Files	YES	NO
Mouse Driven User Interface	YES	NO
Exit Without Restarting Amiga	YES	NO
Runs From Workbench or CLI	YES	NO
Makes Multiple Simultaneous Copies From One Original	YES	NO
Copies Itself	YES	NO
Copies The Mirror	YES	NO
Price	$39.95	$49.95



NOW YOU CAN SAVE ANY SCREEN, FROM ANY PROGRAM, ANYTIME WITH GRABBIT.

With GRABBIT you can capture exactly what you see on your screen in an instant, regardless of what programs you're running. GRABBIT works with all video modes, including "Hold and Modify." What's more, GRABBIT runs completely in the background, transparent to your other software. GRABBIT is always ready for you to use, even when you're in the middle of another program. As if that is not enough, GRABBIT requires only about 10K RAM to operate, and it supports dozens of printers. GRABBIT is truly a productivity power tool for your AMIGA!

GRABBIT is far superior to other screen-printing "programs" because of its small size and quick performance. No complicated setup is required, just install and go! Also, GRABBIT doesn't require the screen to remain visible during printing or saving, and stopping the print operation is as easy as starting it.

GRABBIT supports all standard Amiga printer drivers. GRABBIT also supports full color printing.

In addition to GRABBIT's printing capabilities, the package also includes a powerful utility program "ANYTIME." The ANYTIME bonus program is a "Preferences" style palette requester that allows you to change any colors of any screen, anytime! With ANYTIME, you are NOW capable of customizing all colors to match your printer's hardcopy to the screen's colors.

Once you start using GRABBIT and the bonus program ANYTIME you will want it on every disk. You get all the power of this sizzling new software for an unbelievably low **$29.95** + $5 shipping and handling.

With Key Genie — One Key Launches 1000 Strokes!

This amazing keyboard macro processor is just what you need to give your fingers a rest. The Genie is always at work to save you time and keystrokes. Complicated or repitious keyboard sequences are easily assigned to a key you choose through the Genie's Pop-Up Macro Definition Window. You can also load and save your favorite macro sequences on disk. Once saved, the macros can be automatically installed at startup to save time. In addition to the Genie's powers, Discovery Software has added a bonus program "Turbo-Shell". The Shell is an AmigaDOS performance enhancer that you shouldn't be without! The Shell gives you the capability to recall previous CLI commands with the arrow keys so that mistyped commands can be quickly corrected, and frequently used commands can be easily repeated. Fast AmigaDOS command replacements give you UNIX-style performance from your Amiga.

What other software does so much for you at such a low price. Only **$49.95** + $5 shipping and handling.

When ordering from overseas, add an additional $5.00 shipping for first class airmail.

11-1-86

DISCOVERY SOFTWARE INTERNATIONAL 903 E. Willow Grove Ave., Wyndmoor, PA 19118 (215) 242-4666

make the system "build" the RAM disk.

Assuming you have already used IconEd to create your own personal disk icons, you may be disappointed to see that the RAM disk icon is the old default disk icon. To change this, use the following command in place of the do-nothing Dir command suggested above:

COPY DISK.INFO TO RAM:

This command copies your system disk's icon image into the file Ram:Disk.info, where it will be used to draw the RAM disk's icon. You could instead create a customized icon just for the RAM disk and keep that icon in a separate file on the system disk. If your RAM disk icon were called, say, Ramdisk.info, you would use this startup-sequence command:

COPY RAMDISK.INFO TO RAM:DISK.INFO

Some notes: First, Intuition apparently ignores the extra disk-type info file on the system disk. Next, since the RAM disk icon is created only once each session, there is no need to leave the Disk.Info file in the RAM disk after it's been used. Third, a Snapshot of the RAM disk will not be permanent unless you copy RAM's Disk.Info file back to its file on the system disk. Of course, files in the RAM disk will be lost when you reboot or turn off your Amiga.

Let me mention once again that this trick will only work on version 1.2 system software.

Marcus Brooks
Austin, TX

Hidden Debug Menu

For a bit of fun, type this command from a CLI window: LOADWB –DEBUG. Now go to the Workbench and select the Special menu. While still holding the mouse button, slide the pointer to the right and voila! A hidden menu. The items in this menu cause debugging information to be sent out the serial port at 9600 baud. Anyone with an RS-232 terminal lying around can tap into this feature.

Bryce Nesbitt
Berkeley, CA

Words And Pictures

Notepad Address Book

The Notepad consists of 10 pages, each with 49 rows of text. If each page is titled with 3 consecutive letters of the alphabet, the remaining 48 rows can be used as an address book and/or phone directory. I've found it easiest to keep the screen default width and use the sizing gadget to drag the window to full length before saving. There is even room for an opening title page. If you put V, W, X, Y and Z at the top of the last page, you have enough room for a page of notes or important dates on the last page. Using the Style menu allows for endless creativity, and the address/phone listings can easily be printed.

Paul Elliot
Suffield, CT

Getting Behind Textcraft

Some software, like Textcraft, has no documentation for how to change screens with the Amiga-N and the Amiga-M keystrokes. For instance, in Textcraft, most people are probably selecting the Quit option from the menu whenever they want to see the Workbench screen (e.g., to move an icon for another project into the same drawer as the Textcraft project). But there is a much faster way to return to Workbench without breaking out of Textcraft: Just open the Workbench Clock before opening Textcraft. Then, from within Textcraft, Amiga-N will switch you to the Workbench screen and Amiga-M will return you to Textcraft.

Andrew Lichtman
Nellie Gail Ranch, CA

CLI Typewriter

If you want to use the Amiga and your printer as a typewriter, all you have to do is enter COPY * TO PRT: from the CLI prompt. From then on, everything you type on the keyboard will be printed on the printer. To cancel the command, press CTRL and \ at the same time. We've used this technique to make labels for slide trays, 8mm movies, video cassettes and even a list of Amiga commands for quick reference.

Ms. V. Bende
Whitby, Ontario
Canada

Ed Prompts

If you're like me and you use ED a lot, you also keep forgetting what keys do what. I end up overusing the "easy-to-remember" keys and wasting a lot of time thumbing through the manual. Here is a better way. Using ED (what else?), make a cheat sheet with all the info you think you'll need. Make sure that it doesn't fill up more than one screen and then save it as EdHelp, or something you'll remember. Before using Ed for another project, just enter TYPE ED-HELP. You'll see your screenful of information appear. Next, use ED as usual, and anytime you need some help, simply click the window depth arrangement gadget in the upper right-hand corner of the screen; your list will appear. Click the window gadget again to return to your ED project. The only drawbacks to this technique are that you have to do it for each ED session and the help screen can only take one screen or the info at the top scrolls off.

James Nakakihara
Utsunomiya, Japan

Double-Spaced Printout Fix

Here are a few hints for those of you having printer problems. If you have a printer (like many Radio Shack printers) that requires a CR (carriage return) without the LF (line feed) or it will double space everything, there is a simple way to fix your printouts. First, create a simple two-byte file named HLF (or something appropriate) composed of the control sequence that your printer needs to get a half forward linefeed. (On my Tandy printer this sequence is $1B,$1C.) Then, in the first line of your startup-sequence file add the line TYPE > PAR: HLF or, if you are using the serial port, TYPE SER: HLF.

Now whenever you boot up your system you can output text via the PRT: and have everything come out single spaced, assuming that you are using the Generic printer driver or a driver that works with your printer. If you wish, you can also include any escape sequences in your HLF file to do things like condensed or bold printing.

Scott Cabit
Columbia, CT

Printing With ED

Before buying a word processor I used the Amiga editor, ED. I gather that many people use ED for letters, short articles, etc., so this tip might be of help to them.

I found that using the formula described in your magazine for controlling the printer paper feeds, formatting and page breaks wasn't the answer I needed, so I came up with a one-line alternative. After the text has been drafted, use this ESC command:

T; SL10; SR70; BS; M30; BE; WB/PRT:

In this example you can change the values for left margin (10), right margin (70) and last line to be printed (30) to fit your print- ►

XEROX

The Xerox 4020 Ink-Jet Printer is making headlines with unlimited color.

If you want to make headlines with your next business presentation, try adding color with the Xerox 4020 Color Ink-Jet Printer.

The 4020 is a high-quality, attractively priced printer that mixes text with brilliant color graphics. It employs the latest advances in ink-jet technology to deliver seven distinct colors which combine to produce over 4,000 different shades.

But the 4020 makes more than just a colorful impact. It also has a resolution that few color ink-jet printers can match, prints 2 to 3 times faster than comparably priced color printers, and can be installed by just about anyone in less than 10 minutes.

Like all Xerox products, the reliable 4020 is backed by one of the best service and support organizations in the industry: Team Xerox. To learn more about the 4020 Color Ink-Jet Printer, send in the coupon below or call 1-800-TEAM-XRX, ext. 199A.

Team Xerox

We've got a colorful solution to your business needs.

I'd like to learn more about the 4020 Color Ink-Jet Printer and what it can do for my business presentations.

☐ Please send me more information.
☐ Please have a sales representative contact me.

Send this coupon to: Xerox Corporation, P.O. Box 24, Rochester, NY 14692.

NAME ________ TITLE ________

COMPANY ________

ADDRESS ________ CITY ________

STATE ________ ZIP ________ PHONE ________

Or, if you can't wait, call

1-800-TEAM-XRX, ext. 199A.

199A. (1-800-832-6979, ext. 199A.) 999-5-87

er's needs and page length. To print subsequent pages, replace the T with M values, for example:

M31; SL10; SR70; BS; M60; BE; WB/PRT:

Lee Akazaki
Toronto, Canada

DeluxePaint Keyboard Colors

I use DeluxePaint frequently, and over the months I have stumbled upon a couple of undocumented keyboard color commands.

The 7 and 8 keys (on either the main keyboard or the keypad) subtract and add to the level of red in the currently selected foreground color. The 4 and 5 keys work the same way for green. The 1 and 2 keys work for blue. These are the keyboard equivalents of the RGB sliders on the graphic palette. Note that these keys are not subject to the limitations of the graphic palette involving colors 1 and 2. (See the manual for details.)

Pressing the Help key will set color 1 to black and color 2 to white. These colors are used by DPaint to draw the menus, and loading a picture made with another program may make their values too close, rendering the icons and title bar invisible.

Finally, the braces { and } (shifted bracket keys) allow you to step through the palette to select the background color in the same way the bracket keys [and] (unshifted) allow you to select the foreground color.

Bryan Costin
Woodbine, MD

Disk Management

Copy of Another Type

Another way to copy a file is by using the Type command under the CLI.

The Type command will display the contents of a file in either ASCII or hex, depending upon the option used and, of course, the file being accessed.

Normally, the return is to the screen or prt:. However, I have discovered that it can also be to a disk or file, and if a file is not specified, one is created.

The format is as follows:

TYPE DF?: *file name* TO *file name*

For example, let's say you have a file on a disk in drive 1 called AmigaWorld and you want a copy on the disk in drive 0 to be called Mags. In the CLI, you would type the following command string:

TYPE DF1:AMIGAWORLD DF0:MAGS

Note that the use of TO is optional. When you now do a directory search, you should find a new file called MAGS on DF1:.

David Morgan
Robina, Queensland, Australia

Transferring Preferences

Some Amiga software will not load if your Workbench disk has been loaded before it; therefore, the most obvious way of using the same Preferences settings as on your Workbench disk is to open the Preferences on the new disk and reset everything. However, there is an easier way. Although it doesn't say so anywhere in the manual, everything saved in Preferences resides in the system-configuration file. To easily transfer all your settings to a different disk, just enter the following line from the CLI:

COPY WORKBENCH:DEVS/SYSTEM–CONFIGURATION TO *YourNewDisk VolumeName*:DEVS/SYSTEM–CONFIGURATION

Don't forget to make a backup of any disk that you are writing to before using this technique.

Mark Cashman
Windsor, CT

Three Copy Tips

1) Copying files can be a very slow process, especially if the source and destination files are on the same disk. This is because AmigaDOS uses a maximum buffer size of 512 bytes. 512 bytes are read, then written, read then written, etc., until the file is copied. You can increase the buffer size, and on files of 10K or so you can also increase the copy time by a factor of five or more. From the CLI use this sequence:

COPY *sourcefilename* TO RAM:
COPY RAM:*sourcefilename* TO *destinationfilename*
DELETE RAM:*sourcefilename*

2) If you find yourself copying a large number of files between multiple disks, no doubt you're sick of reinserting your Workbench disk every time you turn around. The reason you have to keep swapping disks is that AmigaDOS has to reload the copy program each time it is used. You can eliminate all the swapping by putting a copy of COPY on a RAM disk by entering the following sequence from CLI:

COPY C:COPY TO RAM:

Then when you want to copy a file type:

RAM:COPY *sourcefilename* TO *destinationfilename*

3) The last copy trick is to use RENAME instead of COPY. This will only work if the source and destination files are on the same disk and if you only want one copy of the file, not two. For example, you could use RENAME to move a file from one subdirectory to another subdirectory on the same disk. The advantage to using RENAME over COPY is that the file is not copied—only its pathname is changed. The RENAME command takes less than a second, regardless of the size of the file.

David Allen
Westbury, NY

More Space on Disks

One way to make more space on disks is to delete the files that are not needed in the SYS:DEVS/PRINTERS directory. If you use one of the printer drivers provided by Preferences, you simply delete the others. Let's say you want to keep the Generic printer driver and delete the rest. Just enter the following from the CLI prompt:

```
CD DEVS
COPY PRINTERS/GENERIC TO SYS:T
CD :
DELETE SYS:DEVS/PRINTERS ALL
CD DEVS
MAKEDIR PRINTERS
COPY SYS:T/GENERIC TO PRINTERS
CD :
```

It is a good idea to do a DIR DEVS/PRINTERS before deleting printer files just to see how they list your printer (type it exactly as they do) and, as always, *don't do this on your original Workbench disk!* What if you buy a new printer someday?

Eric Schmid
Varberg, Sweden

Quicker Disk

With a two-drive system, you can improve your directory access times on frequently used disks using a simple trick. Just format a new disk (and INSTALL if it is going to be a Workbench disk), then with the disk you want "fixed" in the internal drive and the new disk in the external drive, from the

Continued on p. 96

Circle 305 on Reader Service card

Software designed for AMIGA.

Lattice® C Compiler *$225.00*

New version 3.1 of the AMIGA DOS C Compiler replaces version 3.03. Major enhancements include the addition of: TMU, an assembler, a faster linker and version 3 MS-DOS.

With more than 30,000 users worldwide, Lattice C Compilers set the industry standard for MS-DOS software development. Lattice C gives you all you need for development of programs on the AMIGA. Lattice C is a full implementation of Kernighan and Ritchie with the ANSI C extensions and many additional features.

Professional Lattice® C Compiler *$375.00*

A new product called the Professional Lattice C Compiler is now available. It includes the C Compiler package (complete with TMU), plus LMK, LSE and the Metascope Debugger.

AMIGA C Cross Compiler *$500.00*

Allows AMIGA development on your MS-DOS system. Price includes the Professional Lattice C Compiler described above.

Lattice Screen Editor (LSE™) *$100.00*

Designed as a programmer's editor, *Lattice Screen Editor (LSE)* is fast, flexible and easy to learn. *LSE's* multi-window environment provides all the editor functions you need including block moves, pattern searches and "cut and paste." In addition, *LSE* offers special features for programmers such as an error tracking mode and three Assembly Language input modes. You can also create macros or customize keystrokes, menus, and prompts to your style and preferences.

Lattice dBC III™ Library *$150.00*

The *dBC III library* lets you create, access and update files that are compatible with Ashton-Tate's dBASE system. *dBC III's* C functions let you extend existing dBASE applications or allow your users to process their data using *dBC III* or dBASE III.

Lattice Text Utilities (TMU™) *$75.00*

Lattice Text Utilities consists of eight software tools to help you manage your text files. GREP searches files for the specified pattern. DIFF compares two files and lists their differences. EXTRACT creates a list of file names to be extracted from the current directory. BUILD creates batch files from a previously generated file name list. WC displays the number of characters and optionally the checksum of a specified file. ED is a line editor which can utilize output from other *TMU* software in an automated batch mode. SPLAT searches files for a specified character string and replaces every occurrence with a specified string. And FILES lists, copies, erases or removes files or entire directory structures which meet the specified conditions.

Lattice Unicalc® Spreadsheet *$79.95*

Unicalc is a simple-to-operate program that turns your AMIGA computer into an electronic spreadsheet. Using *Unicalc* you can easily create sales reports, expense accounts, balance sheets, or any other reports you had to do manually.

Unicalc offers the versatility you've come to expect from business software, plus the speed and processing power of the AMIGA.

• 8192 row by 256 column processing area • Comprehensive context-sensitive help screens • Cells can contain numeric, algebraic formulas and titles • Foreign language customization for all prompts and messages • Complete library of algebraic and conditional functions • Dual window capabilities • Floating point and scientific notation available • Complete load, save and print capabilities • Unique customization capability for your every application • Full compatibility with other leading spreadsheets • Full menu and mouse support.

Lattice MacLibrary™ *$100.00*

The *Lattice MacLibrary™* is a collection of more than sixty C functions which allow you to quickly and efficiently take advantage of the powerful capabilities of the AMIGA.

Even if your knowledge of the AMIGA is limited, *MacLibrary* can ease your job of implementing screens, windows and gadgets by utilizing the functions, examples and sample programs included with the package.

Other *MacLibrary* routines are functionally compatible with the most widely used Apple® Macintosh™ Quickdraw Routines™, Standard File Package and Toolbox Utility Routines enabling you to rapidly convert your Macintosh programs to run on the AMIGA.

Panel™ *$195.00*

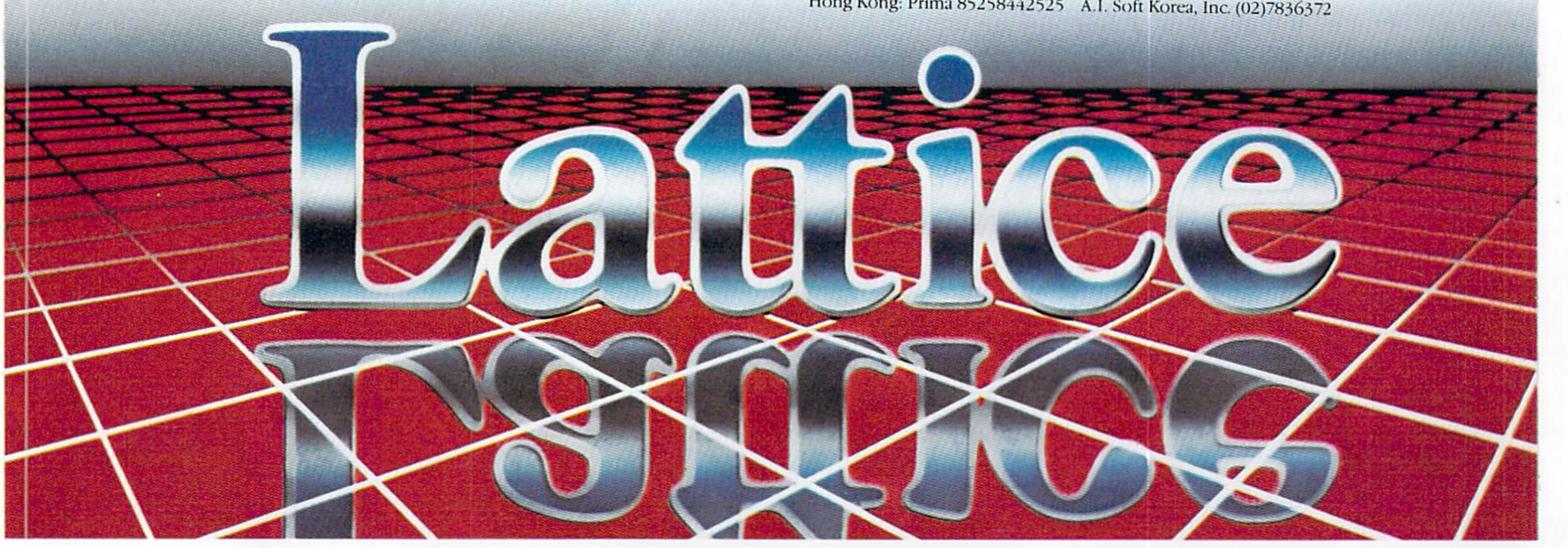
Panel will help you write your screen programs and layer your screen designs with up to ten overlapping images. *Panel's* screen layouts can be assigned to individual windows and may be dynamically loaded from files or compiled into a program. *Panel* will output C source for including in your applications. A monitor and keyboard utility is also included to allow you to customize your applications for other systems.

With Lattice products you get *Lattice Service* including telephone support, notice of new products and enhancements and a 30-day money-back guarantee. Corporate license agreements available.

Lattice

Lattice, Incorporated
Post Office Box 3072
Glen Ellyn, Illinois 60138
(312) 858-7950 TWX 910-291-2190

INTERNATIONAL SALES OFFICES: Benelux: Ines Datacom (32) 2-720-51-61
Japan: Lifeboat, Inc. (03)293-4711 England: Roundhill (0672)54675
France: SFL (1)46-66-11-55 Germany: Pfotenhaur (49)7841/5058
Hong Kong: Prima 85258442525 A.I. Soft Korea, Inc. (02)7836372



Beginner's Guide To the CLI

There's more to life than icons, and the

Command Line Interface isn't as frightening as it sounds.

By Peggy Herrington

Now that you own the world's most powerful personal computer, isn't it about time you learned how to use it? If you're just running programs the easy way from Workbench, you're not getting the most from your system. In fact, you're missing out on all sorts of power that you paid for.

The CLI—the Command Line Interface (or Interpreter, if you prefer)—is your means of communicating directly with AmigaDOS, the Disk Operating System. While using the CLI is trickier than clicking on icons, it taps into the power behind Workbench, and there are many things you simply can't do any other way. But rather than listening to reasons, try a taste of the CLI and decide how palatable it is.

Escape Hatch

Don't worry, nothing you can type from the CLI will damage your system. The worst you can do is inadvertently alter a disk, and since you *always* keep a backup copy, you can start over should something go wrong. You could conceivably meet up with a "guru" or lockup the system, but, if all else fails, you can always do a warm boot: simultaneously press the Control Key (CTRL) and the two red Amiga keys on either side of the space bar. Use the warm boot sparingly—only when there is no other recourse—because a warm boot clears random memory, destroying your hard work in RAM.

Play It Again, Sam

Before using the CLI, you have to activate it. To begin, click on the Preferences Icon in Workbench, and, after the Preferences screen appears, click the CLI ON square. You might want to change the screen output to 80 columns while you're at it. Save the Preferences changes by clicking the Save indicator. Perform a warm boot to engage the 80-column mode, then click on the Workbench System Drawer to get the CLI Icon. Click on it and voila! You're looking at your first CLI window. Move the box to the top of the screen with the drag bar and, using the gadget in the lower right corner, resize it to fill the entire screen.

Now comes the good part. You'll use the CLI to copy your Workbench disk and to personalize your startup sequence.

When you use Diskcopy from Workbench, you must swap disks whether you have one or two disk drives, but by using Diskcopy from the CLI, you can copy disks on a two-drive system without any swapping. As a precaution, make sure the write-protect notch on your Workbench disk is up in the inhibit position, so that you can't write to it accidentally. If you have two drives, put a new, unformatted disk in DF1:, the external drive, and from the CLI type (it doesn't matter a whit whether you use upper or lower case):

```
CD DF0:SYSTEM
DISKCOPY DF0: TO DF1:
```

This tells the Amiga to copy everything from the disk in the internal drive onto the disk in the external drive. Simply type DISKCOPY DF0: if you have a single drive—the difference is that you'll have to swap disks instead of watching the system read from one and write to the other. When you're finished, put the original disk away and warm boot with the new Workbench disk in DF0:. You'll notice that this method of copying gives the copy the same name as the original disk rather than calling it Copy of *Disk name,* as the Workbench-based copy does. Check it out by typing INFO.

3½" Filing Cabinets

Think of your disks as little filing cabinets. Information is stored in files, which are in turn categorized in drawers or directories, just as in standard office filing cabinets. Directories can be nested, one inside another, but to access inner directories, you must work through the outer ones first. In the CLI, a directory is created with the command MAKEDIR DFn: *directory name* (n being drive 0 through 3, or DH0: for a hard drive). Since a disk must be mounted before the system will recognize it, this duality—the ability to access a disk by its volume name or its device name—comes in handy when you change disks.

You can get the system to recognize the presence of a ►

ILLUSTRATED BY GWYN STRAMLER

newly inserted disk or switch back and forth between directories with the CLI command CD, which stands for Current Directory. CD DF0: will take you to the internal drive, CD DF1: to the external drive, or CD *disk name*: to a specific disk, if you know its name. Like file names, directory names with spaces must be enclosed in double quotation marks. For example, CD Joes Disk: won't work; CD "Joes Disk:" will.

Now type CD DF0:. On the next line, type DIR. The names of all the DIRectories on that disk, followed by programs not inside a directory, will be printed to the screen. DIR OPT A lists all of the directories on the current disk followed by the names of the files inside of each. It's helpful for locating a file on a disk when you've forgotten which directory it's in. With any of these commands, screen output can be paused by pressing the space bar and resumed by pressing back space, or permanently halted with CRTL-C.

If a command doesn't work as you think it should, try typing WHY. Although it may be cryptic, you'll get some kind of an explanation.

Notice the directory named System on the Workbench disk. You can see what's inside it by typing

```
CD SYSTEM
LIST
```

The LIST command shows the size of each file in number of bytes, its protection (if any) and the date and time it was saved. The files with .info extenders contain information for Workbench icons that correspond to programs of like names, Diskcopy and Diskcopy.info, for example. Don't confuse .info extenders with the command INFO that lets you see how full a disk is by blocks (512 bytes each). When you're finished looking around the System directory, type CD / to back up one directory and, in this case, get back to the root directory of the Workbench disk. An interesting place to try navigating nested directories is the DEVS directory on this disk. If you get lost, type CD and the system will tell you where you are. CD DF0: will take you back to the top.

File Folders

Disks hold two kinds of files, those containing readable text or ASCII, and those with binary data, which are usually programs. If you're still in the System directory on the Workbench disk, type TYPE DISKCOPY OPT H and you'll see the binary listing of the program Diskcopy. When you've had your fill, hit CRTL-C to stop. If you tell the system to TYPE that file (or any binary file) without specifying Option H, you'll get flashes and symbols; CTRL-C works here as well. The COPY command moves files between disks, including the RAM: disk, and directories. Try COPY DF0:SYSTEM/DISKCOPY TO RAM:. Then go to RAM: and see if it's there

```
CD RAM
DIR
```

Most public domain and many commercial disks contain text files called something like README, which have information or instructions that aren't in the printed documentation. They frequently don't have icons, and without the CLI, you might not even know they're there. Text files can be read from the CLI by simply typing them to the screen: TYPE README, for example, although the name must be entered precisely as it appears on the disk, enclosing it in double quotation marks if it contains a space. Try it yourself by reading the Startup-sequence text file in the s directory of your Workbench disk:

```
CD DF0:S
TYPE STARTUP-SEQUENCE
```

Text files can be sent to a printer with TYPE *filename* TO PRT:. You can also turn the CLI and your printer into a typewriter by typing COPY * TO PRT:. Everything you type after will be printed on paper when you hit return. Hold down CTRL and \ (backslash) to get the CLI back. Incidentally, if a command doesn't work as you think it should, try typing WHY. Although it may be cryptic, you'll get some kind of an explanation. And if you can't remember the proper syntax, type the command followed by a space and a question mark. You then get the syntax of the command, although the results are normally very cryptic.

Say Hi to ED

Many programs used by the CLI are in the C (Command) directory. The system seeks them via a standard PATH (which you can change) and you call them by typing their names. Try typing

```
CD DF0:C
LIST
```

Or, if you're the orderly type, DIR will display them alphabetically.

One of the handiest of these programs is the text editor, ED. While not a full-fledged word processor, ED is convenient for making batch or EXECUTE files, like Startup-sequence in the S directory, which is executed at start-up. This file contains CLI commands that are executed whenever you boot up your system. You can edit this file to ask you for the current date and time and automatically move some oft-used commands into the internal RAM: disk where they will operate faster and free-up your disk drives for more important chores. To call ED and load the file, type:

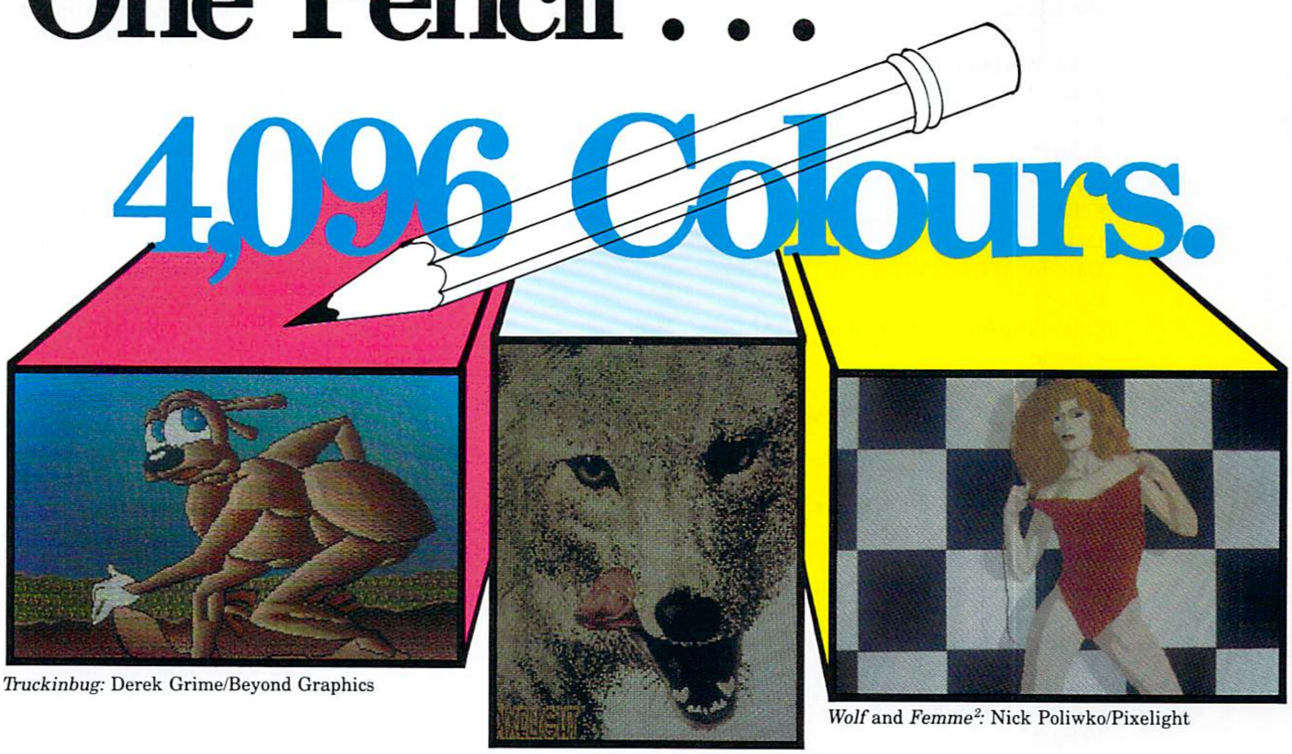
```
CD DF0:
ED S/STARTUP-SEQUENCE
```

ED will present the disk file onscreen. To move about, use the arrow keys or the following combinations:

ESCape-B moves to the bottom of the file.
ESC-T moves to the top of the file.
ESC-Q quits *without* saving any of your changes.
ESC-X exits and rewrites the file, saving your changes.
CTRL-B deletes the line the cursor is on.
CRTL-A adds a blank line below the line the cursor is on. ►

One Pencil . . .

4,096 Colours.



Truckinbug: Derek Grime/Beyond Graphics

Wolf and *Femme²*: Nick Poliwko/Pixelight

EASYL™

Pressure Sensitive Drawing Pad

What It Is . . .

EASYL is a pressure-sensitive tablet which allows you to draw directly into the computer with an ordinary pencil and paper.

How It Works . . .

Draw or trace directly into **Deluxe-Paint II, Pagesetter** or any other 1.2 mouse-based software. Click to load the driver, keystroke to remove it. The Source Code is available.

ANAKIN RESEARCH, INC.
100 Westmore Drive, Unit 11C
Rexdale, Ontario, Canada
M9V 5C3
(416) 744-4246

Who Uses It . . .

Artists, animators, teachers, designers and students around the world have found EASYL to be the ideal drawing tool for the Amiga.

Why Buy It . . .

Because your time is valuable, and you want to express your creative ideas quickly and naturally.

Where To Buy It . . .

Contact your local Amiga dealer.

Coming Soon!
Cel animation/pencil test software and storyboard software.

EASYL is distributed in North America and Europe by:

Canadian Distributor
Air-Stat of Canada, Markham, Ont. — (416) 477-9440, (416) 493-8854

U.S. Distributors
American Software Distributors Inc, Urbana, Ill. — 1-800-225-7941, in Illinois (217) 643-2050
CSS East, Cambridge Mass. — 1-800-343-7535, in Mass. (617) 876-2505
CSS, Chicago, Ill. — 1-800-422-4912, in Illinois 1-800-331-SOFT
Southern Technologies Inc., Dallas, Texas — (214) 247-7373

European Distributors
Easy Computing, Brussels, Belgium — 02 660 6390
Ingeniorfirmaet F. J., Bagsvaerd, Denmark — 02 44 0488
Precision Software Limited, Worcester Park, England — 01 330 7166
Computer Concept, Paris, France — 47 07 5715
Quest, Verona, Italy — 45 585 302
Data Technology Management (DTM), Wiesbaden, West Germany — 06121 407989
Microtron Computer Produkte, Pieterlen, Switzerland — 03 287 2429

Circle 306 on Reader Service card.

When you feel comfortable with the controls, edit the file to read:

```
Echo Workbench Release 1.2 Version 34.37
Echo "Note: This System insured by Smith & Wesson"
Echo " "
Echo "Last Date: "
Date
Echo " "
Date ?
Date to DF0:S/LAST-STARTUP-DATE
Echo " "
Echo "Putting SYStem into RAM:Drive. Hold on a
minute. . ."
Echo " "
Echo "If you don't want me to, press CTRL-D now!"
Wait 5
Makedir RAM:C
Copy SYS:C TO RAM:C ALL QUIET
Assign C: RAM:C
Echo " "
Echo "Press CTRL-D to abort Loading Workbench"
Wait 5
Loadwb
Echo " "
Echo "Let's compute!"
Echo " "
```

When you're done, quit ED with ESC-X, which will automatically save your new file to disk.

Do another warm boot to test your handiwork. The system should prompt you for the time and date in the format HH:MM DD-MMM-YY, recording it in a file called LAST-STARTUP-DATE. Unless you press CTRL-D, the machine will then copy everything from the C directory to RAM: and load Workbench, leaving you an active CLI with Workbench behind it. With this setup, you don't need the Workbench disk in the drive to use most C commands. If you did press CTRL-D and want Workbench, simply type LOADWB. You can customize this file by putting fewer of the C commands into RAM:, where they take up memory and limit your capacity for running large programs. Delete them by first typing

```
ASSIGN C: "WORKBENCH 1.2:C"
CD RAM:C
DELETE #?
```

Walking and Chewing Gum

Everyone knows you run a program from the Workbench by clicking on its icon. From the CLI, you run it by simply typing its location or the path followed by its name, DF1: SCRIBBLE! for instance. If you take the system down the path first by typing DF0: or DF1: and preface the name by typing RUN SCRIBBLE!, you can then multitask. Memory permitting, you can run concurrently as many programs, officially called tasks, as you want, setting the priority for each with SETTASKPRI (−5 to +5). If you have expansion RAM, you can dump a file to your printer with TYPE, create NEWCLIs and SORT another file alphabetically or numerically, use DISKDOCTOR to fix a damaged disk, RUN a background music program and chat online all at the same time.

Still not convinced that using the CLI pays off in the long run? Then consider that many public-domain programs don't come with icon files, which means they aren't accessible from Workbench. You'll need to be fluent in CLI to download and organize them on disks in directories. The FILENOTE command in particular is handy with these files. With it, you can append an 80-character message to a file that will appear when you LIST it. Type FILENOTE *program name* (the note itself, enclosed in quotes if you use spaces).

A host of other commands are accessible from the CLI. They're listed in the Reference Guide bound into this issue. The best way to master them is simply to get in there and experiment. If you get confused, there are a couple of programs on the market that tame the CLI, notably CLI-Mate from Progressive Peripherals, ZING! from Meridian Software and Metacomco Shell from Metacomco. For old fashioned instruction, books on AmigaDOS (listed below) are available, as well. One last tip, if your confidence needs a boost, try typing: PROMPT "%N> Yes, oh Master? ". You'll be pleased with the results.■

Peggy Herrington is a Contributing Editor to AmigaWorld. *Write to her c/o AmigaWorld editorial, 80 Elm Street, Peterborough, NH 03458.*

Software:

CLI-Mate
Progressive Peripherals and Software
464 Kalamath Street
Denver, CO 80204

ZING!
Meridian Software Inc.
PO Box 890408
Houston, TX 77289-0408

The Metacomco Shell
Metacomco
5353E Scotts Valley Drive
Scotts Valley, CA 95066

Books:

The AmigaDOS Manual, 2nd Ed.
Commodore-Amiga/Bantam Books
666 Fifth Avenue
New York, NY 10113

Mastering AmigaDOS
Arrays, Inc.
6711 Valjean Avenue
Van Nuys, CA 91406

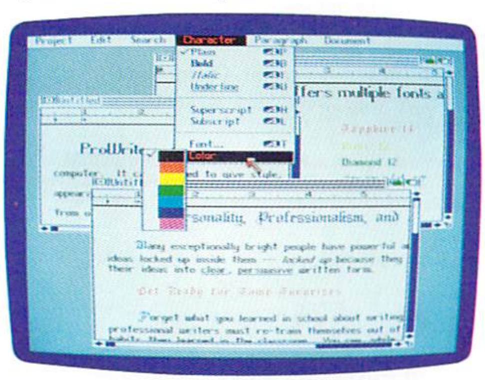
AmigaDOS Reference Guide
Compute! Books
324 West Wendover Ave., Suite 200
Greensboro, NC 27403

Now Look At Word Processing In A Whole New Light.

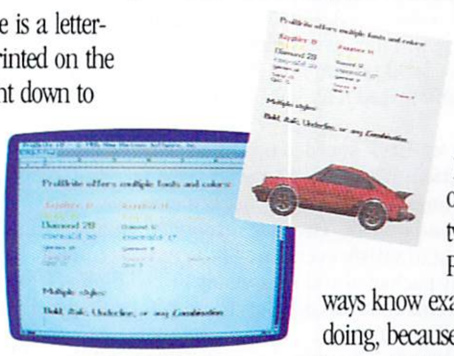
ProWrite.™ All Others Pale By Comparison.

You chose Amiga® because you wanted something more. Now you choose your software for the same reason. Which makes choosing ProWrite word processing a very bright idea.

The Full Spectrum of Capabilities. ProWrite lets you select more than just a typeface: You get proportionally spaced character fonts. Brilliant color. Unsurpassed graphics capabilities. And no surprises. Because what you see on the screen with ProWrite is a letter-perfect picture of what you'll see printed on the page— from header to footer, right down to the last pixel.



Creative Control: A Brilliant Stroke. With six pull-down menus, just a click of the mouse or a few keystrokes is all you need to create, delete, copy, alter, move and otherwise manipulate text. Multiple selections create a limitless choice of character fonts, colors, sizes and styles. Customize everything from a daily diary to an annual report, including color graphics. And with headers and footers always clearly displayed, ProWrite makes even advanced formatting as easy as dragging a mouse.



Ease of Use: Another Bright Idea. ProWrite delivers more of what you bought Amiga for: Professional output. Multi-tasking power. A "creative edge." You can open up to eight windows at once—and perform a variety of editing tasks between them. With ProWrite, you'll always know exactly what you're doing, because you see it done—in living color—right before your eyes.

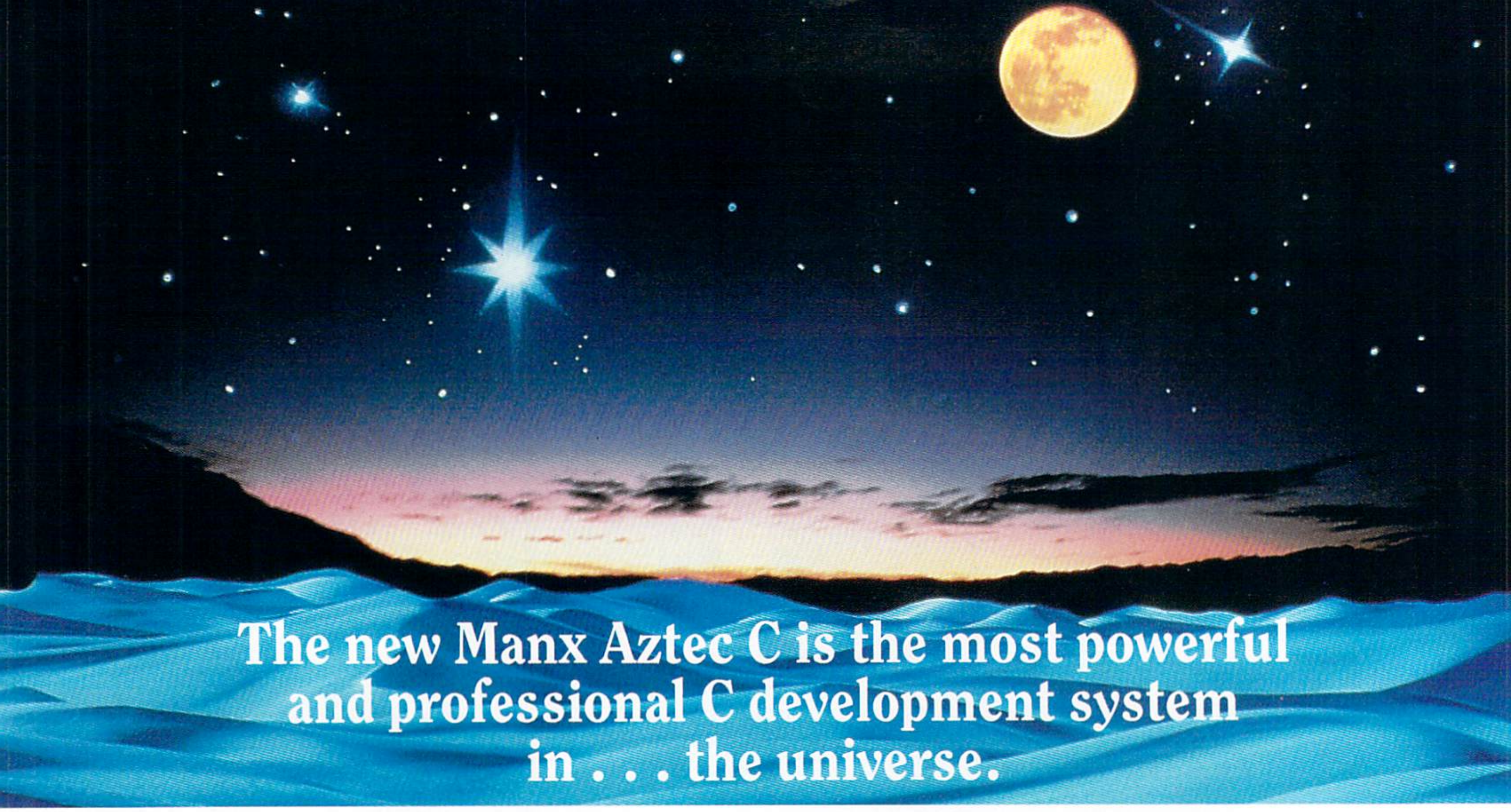
See for yourself what you get with ProWrite. Then, just for fun, take a look at the others—and watch them pale by comparison.

New Horizons
SOFTWARE

First In Personal Productivity And Creativity.

New Horizons Software, Inc. PO Box 43167, Austin, TX 78745 (512) 329-6215

Circle 312 on Reader Service card.



Reviews of Aztec C68k/am v 3.2 were great.

Aztec C68k goes into overdrive for benchmark speed . . .

Byte November, 1986

Aztec C is more than just a C compiler . . . compiles programs significantly faster than Lattice . . . options that for many applications, will improve program execution performance dramatically . . . code generated by Aztec C was dramatically smaller than that produced by Lattice.

Amiga World November, 1986

Now, the New Aztec C68k version 3.4 is even faster, more flexible, and dramatically less expensive.

We did it! We moved all of the great development utilities from Aztec C68k/am-c, our $499 system, to Aztec C68k/am-d, our $299 system. We packaged the same compiler, assembler, linker, libraries, and librarian that are in the $499 system into Aztec C68k/am-p, The Professional System, and priced it at $199. Is that all we did? Not by a long shot!

We also added new features, speeded up the already impressive performance, and we listened. We listened to what users were saying on the technical support lines, on the East Coast Bulletin Board, the West Coast Bulletin Board, and on BIX. Then we responded by correcting, adding and augmenting to satisfy every one of the major requests. The result is three carefully packaged and reasonably priced high performance development systems that will do everything you could possibly want with the Commodore Amiga.

To back up this claim, we offer a 30-day money back satisfaction guarantee. The system must be purchased by an end user directly from Manx to qualify for the 30-day return, but that's it. Less than one per cent of the systems we sell to end users are ever returned, and over 60% of the systems we sell are repeat sales or by recommendation of an existing user. That says a lot.

Manx has a reputation as the expensive system to use for high demand professional applications, but the new prices and reconfigurations make our high powered systems attractive to everyone. In addition, Manx has aggressive site licensing, multiple purchase discounts, OEM, educational, and other special purpose discounts. Call us today for more details. Join the over 50,000 satisfied end users who understand the Manx difference.

New Features: **shorter development times • smaller faster code • support for 1.1 and 1.2 Amiga DOS • direct link of Amiga object modules and libraries • standard driver support • C and assembler support for 68020 and 68881 • scatter loading and support of overlays by segmentation • 1.2 Amiga library function support • debug utilities • enumerated data types • stack depth checking code • Amiga assembly directives support • four floating point formats: Motorola fast float, Amiga IEEE double, Manx IEEE double, 68881 • dynamic memory allocation and window sizing for the Z editor • "touch" command • improved console window RAW and CON switching • stdio I/O is buffered to console • 68010, 68020 debug support • "exec" function supports Amiga PATH command • VT100 emulator (source) • lots of examples**

New Configurations and Pricing

Aztec C68k/am-p (The Professional) — $199
Optimized C, Assembler, Linker, Librarian, Libraries, Examples

Aztec C68k/am-d (The Developer) — $299
Optimized C, Assembler, Linker, Librarian, Libraries, Debugger, Make, Diff, Grep, Support Utilities, Examples

Aztec C68k/am-c (The Commercial System) — $499
Library Source, One Year Of Updates, Z (vi) editor, Optimized C, Assembler, Linker, Librarian, Libraries, Debugger, Make, Diff, Grep, Support Utilities, Examples

All systems are upgradable by paying the difference in price + $10. C.O.D., VISA, MasterCard, American Express, wire (domestic or international), and terms are available. One and Two Day delivery available for all domestic and most international destinations.

To order, or, for information call today.

1-800-221-0440

Telex: 4995812

In NJ or outside the USA call (201) 542-2121.

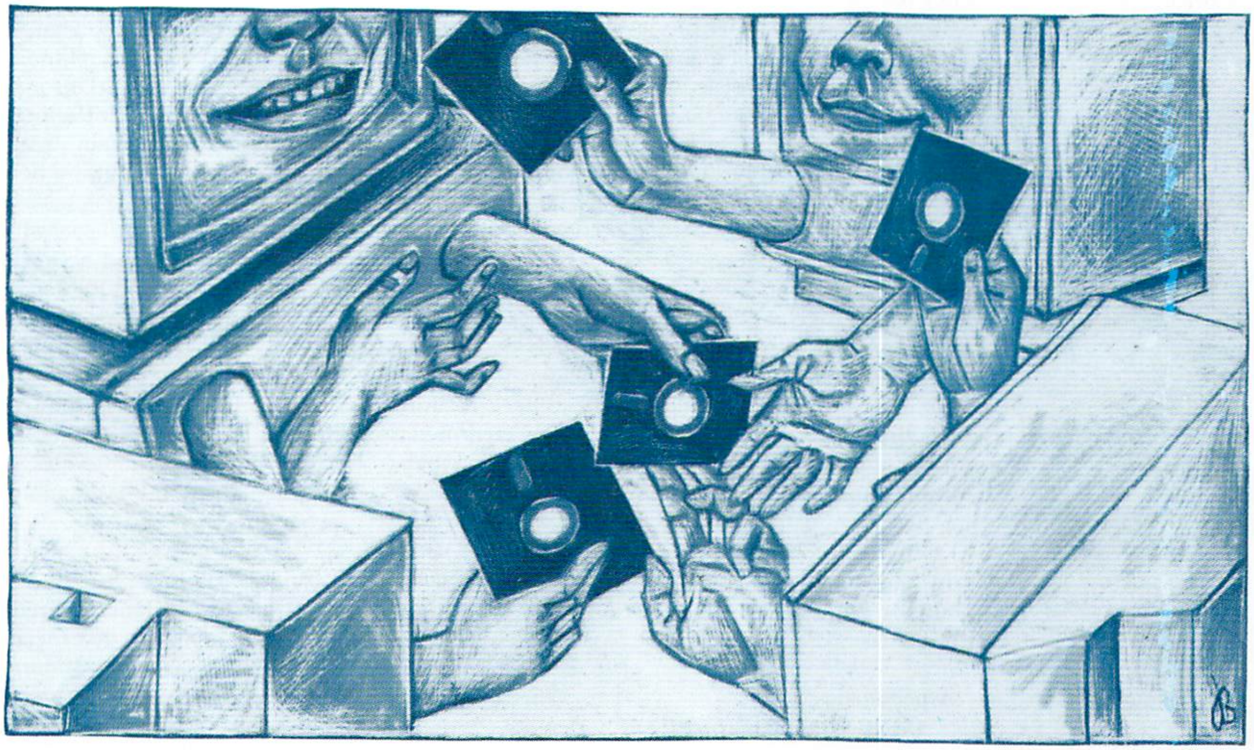
MANX

Manx Software Systems
One Industrial Way
Eatontown, NJ 07724

IFF: The Standard Of Sharing

Though IFF is a buzzword in the Amiga community, for many its meaning remains a mystery. Here's an explanation of what it is and what it's used for.

By Louis R. Wallace



The Interchange File Format (IFF) is a standard method of recording data objects so they can be shared by many different programs. The IFF standard was adopted by Commodore-Amiga soon after the Amiga's release and has become so widely accepted in the Amiga community that any program that does not support it will instantly find itself subjected to an avalanche of criticism. The data can be graphics, text, fonts, music, sound effects, musical instruments or animation. It was developed by Electronic Arts in 1985 to simplify the software development process. While designed with the Amiga in mind, it is not system specific and can be used by other computer systems. In fact, there is growing support for IFF standards to be incorporated into the Atari ST programming community.

By using software that has the IFF format, you have the ability to move data created with one application into another independently-developed application. The beauty of this is most readily apparent to users of the Amiga's many graphic programs. For example, DeluxePaint will easily load and edit a picture created with Aegis Images or Graphicraft. These same images can then be used in DeluxeVideo or Aegis Animator, or transferred to DeluxePrint. With the inexpensive Digi-View graphics digitizer from NewTek, you can take color photographs and save them in IFF format, load them in DeluxePaint for editing and then use them as background scenes for the Animator.

In designing the IFF standard, Electronic Arts considered the history of data standards and types for various microcomputers. One of the systems often cited was the Macintosh, whose Resource Manager and Scrap Manager allow data objects to be swapped between applications. Four-digit identifiers within the file structure ▶

ILLUSTRATED BY LYNN BUSHMAN

represent the different data types in the file. The Macintosh PICT graphics format stores graphics commands and their drawing sequences, which later can be easily accessed and used.

Used by the Apple LaserWriter to describe an image for high-quality printed output, PostScript was examined as a general way to represent a print image. Other file formats studied were the DIF format for spreadsheet data interchange, and InterScript from Xerox, used to transfer text information between different text editors with varying degrees of formatting features support.

By proposing a standard format structure, Electronic Arts wanted to avoid a situation where hordes of programmers would wantonly create whatever file formats were most expedient for their specific applications. They felt "a little behind-the-scenes conversion when programs read and write files is far better than explicit conversion utilities for highly specialized formats." IFF provides the backstage magic.

Since Amiga data can be quite complex with bitmaps, flags, instruments and such, it is organized into chunks. All IFF files contain header information that describes the data's various parts. The data can be interpreted as eight-bit integers, signed or unsigned 16-bit integers and 32-bit integers, depending upon what is required. The data chunks are identified by four ASCII characters in the range of \$20–\$7E. Every IFF file will contain one or more of these four-digit type ID codes. To access any specific chunk of data, the file-reader program examines and discards chunks it has no need of, until it finds the proper one. Since each chunk is identified with its four-digit type ID, it is easy to add new types without interfering with compatibility by simply assigning the new data chunk a unique type ID.

A chunk, the basic building block of IFF files, contains the chunk name or type ID, the size of the data chunk in bytes, and the data itself. In C the structure is defined as:

```
typedef struct  {
  LONG chunkID;
  LONG chunksize;
  UBYTE chunkdata[chunksize];
  } Chunk;
```

If chunksize is an odd number, the chunk will contain a zero pad byte at the end of the chunk data. This byte is not included in the chunksize. The chunk header and the chunksize bytes are not counted either, so the total physical size of a chunk is chunksize + 8 (+ 1 if chunksize is an uneven number of bytes).

All IFF files begin with a special ID type called FORM, LIST or CAT. The CAT type is a concatenation of objects, while LIST is a list of FORMs and CATs. In FORM, the most common type, the four letters (FORM) are found at the beginning of the datafile and imply that the chunk contains an ID, and possibly contains other nested chunks of data. So the letters FORM are the chunk ID, and are followed by a 32-bit integer containing the chunksize. This is the size of the entire datafile chunk itself, including other chunks of data that can be found within its structure. Electronic Arts suggests the first chunksize be used to determine the virtual EOF (End Of File). Next we find another 32-bit integer called a FormType, which defines the type of data structure you will be working with. If the formtype is ILBM (InterLeaved BitMap image with color map), the graphic bitmap image will be defined in the following data. If it is SMUS you have a Simple MUsic Score, while FTXT stands for Formatted TeXT.

All of the IFF files examined here are of the FORM type. By understanding what the file structure is you will be able to use these files within your own programs, or write IFF-compatible files from your programs.

Bitmap Graphics

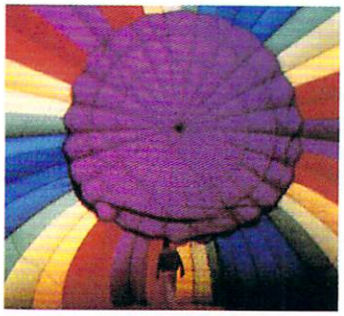
An ILBM file has one of three main functions. It can be a stand-alone image that specifies exactly how to display itself, in terms of resolution, size and colors. It can be an image meant to be merged into a larger picture that has its own number of bitplanes and color palette. Or finally, it can be an empty image file that contains color palette information. Exactly which version it is depends on what it's to be used for.

Figure 1 is the beginning of the contents of an IFF graphics image file. By using the CLI command TYPE FILENAME OPT H, you can display both a hexidecimal data dump and an ASCII listing of any file. The various type IDs are visible in the ASCII portion, while the data itself is available in the hex dump.

As you can see, the first four bytes identify this as an IFF file of type FORM. There are four bytes identifying the chunksize (\$9100 bytes) of the entire file. This is followed by the type ID ILBM, signifying it is a raster-graphics image. The fourth set of four bytes is BMHD (BitMapHeaDer). A PROPERTY type ID, it is a special chunk containing the description of the image, necessary to decode the BODY to come. Following this BMHD chunk ID is the required chunksize information for this PROPERTY chunk, in this case \$14 (decimal 20) ►

Figure 1. Hex and ASCII dump of an Interleaved Bitmap file.

```
FORM ILBM
0000: 464F524D 00009100 494C424D 424D4844    FORM....ILBMBMHD
0010: 00000014 014000C8 00000000 05020100    .....@..........
0020: 00000A0B 014000C8 434D4150 00000060    .....@..CMAP...`
0030: 101010F0 E0C0F0D0 B0C0B0A0 D0B090B0    ................
0040: A090A090 80807060 F0D0B0F0 C0A0E0B0    ......p`........
0050: 90D0A080 C09070B0 8060A070 50906040    ......p..`.pP.`@
0060: D09080B0 7070E0B0 80D0A070 C09060B0    ....pp.....p..`.
0070: 8050A070 40705030 50403040 30203020    .P.p@pP0P@0@0 0
0080: 20202020 7090C060 70904050 60203030        p..`p.@P` 00
0090: 43524E47 00000008 00000000 0001141F    CRNG............
00A0: 43524E47 00000008 00000AAA 00010307    CRNG............
00B0: 43524E47 00000008 00000AAA 00010000    CRNG............
00C0: 43524E47 00000008 00000AAA 00010000    CRNG............
00D0: 424F4459 00009030 240007FE 70F04000    BODY...0$...p.@.
00Eo: 08FE8009 0000FE00 11FFC003 40718D06    ............@q..
00F0: 20CBF7CF C3FF2CFF FFEC0380 41CCFE00     .....,.....A...
```



Actual unretouched photos

DIGI-VIEW

brings the world into your Amiga™!

With **Digi-View** and a video camera, your **Amiga** can see! Faces, logos, artwork . . . anything you can imagine! Simply point your camera and click the mouse. In seconds, whatever the camera sees is painlessly transformed into a computer image that can be printed, stored on disk, or transferred to other programs. Imagine how quickly and easily you can generate stunning video art and animation when you start with high quality digitized photographs or artwork.

Sophisticated software included with **Digi-View** makes it easy to produce dazzling, broadcast-quality color images. Intuitive, on-screen controls are as easy to use as the knobs on your T.V. set. **Digi-View** can capture images in several modes, including 320x200 pixels with up to 4096 colors on screen ("hold-and-modify" mode), and the incredibly detailed 640x400 high resolution mode.

*The key to **Digi-View's** incredible color resolution is this color separation filter which attaches to your black-and-white or color video camera.* *

- IFF disk format works with Digi-Paint™, DeluxePaint™, DeluxeVideo™, DeluxePrint, Aegis Images™, Aegis Animator, and more!
- Saves time! No more hours of freehand drawing and redrawing.
- Send photos over the telephone with your modem and terminal software.
- Capture images for scientific image processing or pattern recognition.
- Spice up business graphics — slide show program included.
- Incorporate photos in posters and greeting cards.
- Use **Digi-View** pictures in your BASIC programs.
- Catalog images with IFF database programs.
- Make red/blue 3D photos.
- A powerful tool for commercial graphic artists!

Panasonic WV-1410 video camera w/lens $280
CS-1L Copy stand w/lights .... $ 75

Only $199.95

includes video digitizer module, color separation filter, software and manual.

Orders Only 1-800-843-8934
Customer Service (913) 354-9332

NEWTEK
INCORPORATED
701 Jackson • Suite B3 • Topeka, KS • 66603

Amiga is a trademark of Commodore-Amiga, Inc. Digi-View and Digi-Paint are trademarks of NewTek, Inc. DeluxePaint, DeluxeVideo, and DeluxePrint are trademarks of Electronic Arts, Inc. Aegis Images and Aegis Animator are trademarks of Aegis Development, Inc.

* Digi-View software version 2.0 (or newer) required to use color camera. For maximum resolution use monochrome camera with 2.1 interlace. High-res color modes require 1 Meg expansion RAM.

Circle 317 on Reader Service card.

bytes. The next 20 bytes are the most descriptive parts of the ILBM, and are best understood in the context of a C chunk structure.

```
typedef struct  {
   LONG chunkID;
   LONG chunksize;
   UWORD width,height;
   WORD xposition,yposition;
   UBYTE numberofplanes;
   UBYTE mask;
   UBYTE compression;
   UBYTE padbyte;
   UWORD transparentcolor;
   UBYTE xAspect, yAspect;
   WORD pageWidth, pageHeight;
     } BitMapHeader;
```

You will see a one-to-one correspondence between the C structure above and the 20 bytes of data.

If you look at the data in Figure 1 following the BMHD chunksize word ($14), you will see a one-to-one correspondence between the C structure above and the 20 bytes of data. The structure shows that two 16-bit unsigned words of two bytes each contain the values for the width and height of the image. In this case these are $0140 (320 decimal) and $00C8 (200 decimal). The next two words are the x and y position of the image on the screen. Here they are both zero, but if this had been a BOB or sprite, a different position might have been required.

The next byte in the structure is the number of bitplanes deep the image is. It can be from one to five (six in Hold-and-Modify mode) planes deep. The sample image is five bitplanes. Byte 10 is the MASK flag, indicating which type of four possible masking types is being used. The first possibility is mskNone with a value of zero, meaning there is no mask and the image is an opaque rectangle. A value of one indicates mskHasMask, signifying there is an interleaved mask plane in the BODY bitplane data. A value of two is mskHasTransparentColor, indicating some portions of the image are transparent. The value of three, mskLasso, means the file reader can construct a lasso mask in the manner of MacPaint. The example in Figure 1 has the value two, so the image has the MASK mskHasTransparentColor.

Byte 11 in the BMHD PROPERTY chunk is the compression flag. The value zero means no compression. The example has the value one, signifying the bitmap data has been compressed. A value of one further indicates it is the standard compression type used by Electronic Arts for raster graphics. If the compression flag is greater than one, it signals a different compression routine, which must be known to any reader program in order to decode the data. Electronic Arts has released into the public domain standard C procedures, called Packer and Unpacker, to compress and uncompress ILBM data. (For an Amiga Basic IFF ILBM decompression routine, called BRUSHESTOBOBS, see *AmigaWorld* Jan/Feb 1987.) Byte 12 is a pad byte of zero and is not used at the moment in the BMHD data, but could be used in the future.

The next two bytes are the transparentColor value, which specifies which bit pattern is to be transparent, and are relevent only if the mask flag is mskHasTransparentColor or mskLasso. For the example, the value is zero.

Bytes 15 and 16 are the xAspect and yAspect ratios. In most cases these will be 10:11 ($0A:$0B). Finally, the words pageWidth and pageHeight are the size in pixels of the source page, in this case 320 × 200 ($140, $C8).

The next type ID, CMAP (Color MAP), is a PROPERTY chunk structure that contains the information for the color registers. Its C structure is:

```
typedef struct  {
   LONG chunkID;
   LONG chunksize;
   UBYTE chunkdata[chunksize];
     } colormap;
```

The chunkID is CMAP, the chunksize is 3×2^ numberofplanes (96 decimal in this case), and the chunkdata is the RGB data values for each color, expressed as triplets. The triplet (0,0,0) is black, while (255,255,255) is white. Only the high-order bits are used on the Amiga, so each shade is limited to 16 increments (16 × 16 × 16 = 4,096).

CRNG (Color Register RaNGe) is an ID type used by DeluxePaint from Electronic Arts to identify a continuous range of color registers for a shade range or color cycling. Its chunk structure is very simple.

```
typedef struct  {
   LONG chunkID;
   LONG chunksize;
   WORD pad1;
   WORD rate;
   WORD active;
   UBYTE low, high;
     } CRange;
```

Here pad1 is two unused bytes, set to zero. Rate is the color-cycle rate, active indicates whether color cycling is on (one) or off (zero). Low and high bytes are the color registers used in this color range.

The BODY type ID is a standard chunk structure consisting of a chunkid, chunksize and chunkdata, followed by a pad byte if needed. It represents the actual bitmap image.

The Amiga's raster images are *interleaved* in the ILBM file. Figure 1 describes a five-bitplane image, so the data consists of one scan line from each bitplane before going to the next scan line. The length of the scan line must always end on a word (two-byte) boundary; if not, it is padded with zero bits until it does. This data may then be compressed as in the example.

Other PROPERTY chunk types can be in an ILBM form. There is a GRAB chunk, which has two words identifying the position of a handle, or hotspot, within an image. The DEST chunk tells how to scatter bitplanes into a deeper destination image. SPRT indicates that the image is a sprite. CAMG is a proprietary ►

DIGI PAINT

all at one time!

The 4096 Color ^ Paint Program for the Amiga™

Create original art with a palette of 4096 colors.

*Load H.A.M. images from **Digi-View™** or 32 color images from DeluxePaint™ and other IFF programs.*

From the creators of **Digi-View** comes **Digi-Paint**, the first paint program to take full advantage of the **Amiga's** exclusive "hold-and-modify" mode. No longer are you limited to 32 colors. With **Digi-Paint**, you can use all 4096 colors on screen simultaneously. Features include brushes, smooth shading, magnify, cut & paste, output to printer, and full IFF load and save. **Digi-Paint** was programmed completely in assembly language for the fastest possible response. Give your **Amiga** the graphics power of systems costing thousands of dollars more. See your **Amiga** dealer today or call toll-free for **Digi-Paint**, the 4096 color paint program.

Only $59.95

Orders Only 1-800-843-8934
Customer Service (913) 354-9332

NewTek
INCORPORATED

701 Jackson • Suite B3 • Topeka, KS • 66603

Circle 335 on Reader Service card.

Amiga chunk, for Amiga-only display modes like Hold-and-Modify or Dual Playfield.

Sound

The IFF sound file 8SVX FORM is the defined file format for digitally sampled sound consisting of eight-bit samples. It can be a single (oneshot) octave sound or several octaves. Figure 2 shows a portion of the data obtained from a oneshot IFF sample made with the FutureSound sound digitizer from Applied Visions.

It begins with the chunkID FORM, followed by the chunksize $0BF8. The next four bytes spell 8SVX, indicating an eight-bit digitally sampled sound file. This is immediately followed by the ID VHDR (Voice HeaDeR) chunk structure.

```
typedef struct  {
  LONG chunkID;
  LONG chunksize;
  ULONG oneShotHiSamples,
        repeatHiSamples,
        samplesPerHiCycle;
  UWORD samplesPerSec;
  UBYTE ctOctave,
        sCompression;
  LONG volume;
            }Voice8Header;
```

The fields oneShotHiSamples and repeatHiSamples contain the number of audio samples in the two parts of the highest frequency octave. One can be empty (zero), and in the example oneShotHiSamples contains the amount $BD0, and the repeatHiSamples is zero. The samplesPerHiCycle field contains the number of samples per cycle in the highest frequency, or a zero if that is unknown, as in Figure 2.

The samplesPerSec field contains the sound sampling rate; here it is $2710. Next is ctOctave, the number of octaves in the BODY chunk structure. Since the example is a oneshot octave, it contains the number one. The sCompression field indicates whether the BODY data is compressed or not, and if so what kind of compression has been applied. A zero indicates no compression, as in the example. A one in this field indicates the standard compression method for sampled sound, Fibonacci-Delta encoding, was used. Any other value signifies a different compression technique. However, the Fibonacci-Delta compression technique is not recommended for sampled sound as it may introduce distortion. The last parameter in the VHDR chunk structure is the volume field, a 32-bit integer value.

Though not used in this particular case, there are other chunks that can be included within the 8SVX FORM. Text chunks are used to include specific information, such as NAME, (c) (copyright), AUTH (author) and ANNO (annotations) chunk types. Their chunk structures are all similar.

```
typedef struct  {
  LONG chunkID;
  LONG chunksize;
  CHAR text[chunksize];
    } textchunk;
```

Optional data chunks include ATAK and RLSE, for use in describing an Attack/Sustain/Release envelope.

The BODY data chunk consists of the chunkID, chunksize and the chunkdata. The chunkdata is in signed bytes (−128 through 127 instead of zero to 255), unless it has been compressed. If compressed, the data is simply unsigned bytes of information, until decompression when it becomes signed again.

Another sound FORMTYPE is SMUS (Simple MUsical Score). This is a data format for moving classical (not free-form) musical scores between programs. SMUS uses Common Music Notation (half notes, dotted quarter rests, etc.), and can define which instruments play which notes. It consists of various data-chunk structures, as well as text structures. For example, it begins with the required property SHDR.

```
typedef struct  {
  LONG chunkID;
  LONG chunksize;
  UWORD tempo;
  UBYTE volume;
  UBYTE ctTrack;
    } SScoreHeader;
```

Tempo is expressed in 128th's of a quarter note per minute. Volume is from 0–127, and ctTrack is the number of TRAK (Track) data chunks, up to 255.

The optional property INS1 identifies the instruments used in the score. As with many of the IFF chunks, they can be accepted or ignored by your IFF reader program.

```
typedef struct  {
  LONG chunkID;
  LONG chunksize;
  UBYTE register;
  UBYTE type;
  UBYTE data1, data2;
  CHAR name[];
    } RefInstrument;
```

The register field, from zero to 255, is used to indi- ►

Figure 2. Hex and ASCII dump of an eight-bit sampled sound file.

```
FORM 8SVX
0000: 464F524D 00000BF8 38535658 56484452    FORM....8SVXVHDR
0010: 00000014 00000BD0 00000000 00000000    ................
0020: 27100100 00010000 424F4459 00000BD0    ........BODY....
0030: 00000000 00000000 00010000 00000000    ................
0040: 00000101 01010101 01010101 02010101    ................
0050: 01010102 01010101 01010101 02010101    ................
0060: 01010101 01010101 01000100 01000000    ................
0070: 00000000 00000000 00010101 01010100    ................
0080: 0000FFFF FFFFFFFF FEFDFDFD FDFDFDFD    ................
0090: FEFEFEFF FF000101 01010201 02020202    ................
00A0: 02020202 02030303 03030303 04040404    ................
00B0: 04040404 03030303 03030303 03020202    ................
```

the new MaxiPlan Plus. . .

it's making every other spreadsheet old fashioned!

The original MaxiPlan™ was named the Best Amiga™ Spreadsheet of 1986 by F.A.U.G., the world's largest and most active Amiga user group. Now in 1987, Oxxi is proud to introduce **MaxiPlan Plus™**—the most advanced Amiga spreadsheet ever. With even more time-saving innovations than the award-winning MaxiPlan, the new MaxiPlan Plus includes Microsoft Excel™-like Macros and Utilities.

With MaxiPlan Plus and your Amiga you can:

- Open multiple spreadsheets and graphs
- "Link" data from any number of spreadsheets
- Create a self-running demo or interactive multiple choice quiz, incorporating files from word processors and paint programs
- Automatically create reports such as invoices and purchase orders
- Instruct a data entry person with speech or written prompts
- "Recite" your data entries when checking data accuracy against source documents
- Export graphs via an IFF file to any Amiga paint program

The MaxiPlan Plus Spreadsheet features:

- 512 columns by 65,530 rows
- Function key commands
- Ranges or cells referenced by "Name" or cell numbers
- Written or spoken notes that attach to cells or ranges
- Password protected cells
- Easy-to-use pie, bar, line and area graphs
- Up to 8 graphs per spreadsheet
- Lotus 1-2-3 compatibility
- 63 built-in functions such as:
  - —Financial (IRR, PV, NPV, FV, PMT)
  - —Data base (Index, Find, Lookup)
  - —Presentation (Color, Style, Speech)

With the MaxiPlan Plus Data Base you can:

- Sort on any number of fields in any order
- Maintain up to 16 data bases per spreadsheet
- Create merge files for labels or form letters
- "Find" or "Lookup" any specific record or records

With the MaxiPlan Plus Macros you can:

- Define up to 32 macros for any spreadsheet
- Activate macros with a single key stroke
- Automatically generate macros under record mode
- Use over 70 different macro commands
- Create templates for data entry
- Incorporate speech to instruct, remind or inform user

MaxiPlan Plus™
\$199.00
at your local Amiga software dealer.

Oxxi Inc.
1835-A/Dawns Way
Fullerton, CA 92631
(714) 999-6710

MaxiPlan
Named the best Amiga spreadsheet of 1986 for its outstanding mouse interfaces, unique pull-down menus and advanced speech capabilities.

Excel is a trademark of Microsoft Corp.

Circle 318 on Reader Service card.

cate the instrument to play. If the type field is zero, then it finds the instrument by the supplied name and sets data1 and data2 to zero. If type is one, it uses the instrument on MIDI channel number data1, preset number data2. The NAME[] field holds the instrument name.

TRAK chunks contain the main part of the score. These represent parallel tracks, with one chunk per track. As mentioned, you can have up to 255 TRAKs in a SMUS FORM, but the numerical order of the various TRAKs is also their priority number. So if a reader program is designed to read four TRAKs, it should read the first four and ignore the rest (if any).

The TRAK chunks contain the data in 16-bit words or events that represent commands. These commands can be simple events like "note" or "rest," or more complex like "set instrument." Events are called SEvent records (Simple musical Event), and the 16 bits in each field are broken into two 8-bit parts. The first part is the sID, which determines how the data is interpreted, and the second is the data itself. There are a number of predefined sID values, indicating chords, notes, rests and instruments.

You can understand and use files written by other programmers for your own purposes.

Text

The IFF FORM for text files is called FTXT (Formatted TeXT), and is meant to serve as a means of communication between text applications. These applications can be as simple as printable alphanumeric characters with spaces and linefeeds, or as complex as documents with embedded format information such as fonts, text size or typeface. As with any IFF reader program, FTXT programs can use what they need or understand, while ignoring the rest.

FTXT uses the eight-bit character streams defined in the ISO/ANSI standard (International Organization for Standardization (ISO) and American National Standards Institute (ANSI)). The data is stored in two conventional data chunks, CHRS and FONS.

The FTXT formtype may contain many CHRS (CHaRacter Streams) chunks. Their format in C is:

```
typedef struct {
  LONG chunkID;
  LONG chunksize;
  CHAR text[chunksize];
    } characterstream;
```

The FONS type is a property type that contains information about assigning a particular font to a font register, which can be referenced within CHRS chunks. Obviously, it also should contain information on the font type.

```
typedef struct {
  LONG chunkID;
  LONG chunksize;
  UBYTE fontID;
  UBYTE pad1byte;
  UBYTE proportional_flag;
  UBYTE serif_flag;
  CHAR fontname[];
    } FontSpecifier;
```

The fontID field contains a number from zero to nine. The pad1byte is an unused field, and should be zero. The proportional_flag specifies if the font is proportional or not. The serif_flag indicates if it is a serif or a sans serif font. In both cases, one stands for no, two for yes and zero for an unknown status.

IFF Programming

For Amiga programmers, it is welcome news that these highly defined file formats exist. They more than make up for their occasional extra programming burden in the ease with which you can understand and use files written by other programmers for your own purposes. It is even better to find that Electronic Arts and Commodore have made available to the public domain a large number of C and Amiga Basic programs and C include files that represent and demonstrate the IFF standard.

For $20, the "EA IFF 85" STANDARD FOR INTERCHANGE FORMAT FILES is available from Commodore. An extensive and invaluable manual on the various IFF formats, it provides a disk complete with all the C source routines, include, object, link and text files. There is also the C source to the SEEILBM program, written by Carolyn Scheppner of Commodore's Technical Support Staff. The C source routines include the Packer and Unpacker data compression procedures, as well as file read and write routines.

If you are interested in programming IFF examples from Amiga Basic, the new Version 1.2 Enhancement release contains several useful and educational programs that are also written by Carolyn Scheppner. Besides programs to load, save and print ILBM files from Amiga Basic, there is a routine for converting an ILBM file to the ACBM (Amiga Contiguous BitMap) picture format. While this is not a standard interleaved bitmap file, it does decrease significantly the already fast load time for displaying a picture from Basic.

One of the best utilities to play digitized sounds from Amiga Basic is available from Applied Visions Inc. They have some very interesting examples of playing digitized sounds from Amiga Basic using a special set of library commands supplied with the FutureSound digitizer.■

The Electronic Arts IFF 85 Manual ($20)
Amiga Technical Support
Kim Montgomery
1200 Wilson Drive
West Chester, PA 19380
(Make the check out to Commodore Business Machines and request the IFF docs and disk.)

FutureSound and Amiga Basic digital sound routines ($175)
Applied Visions Inc.
One Kendall Square, Suite 2200
Cambridge, MA 02139
617/494-5417

The Masquerade Is Over

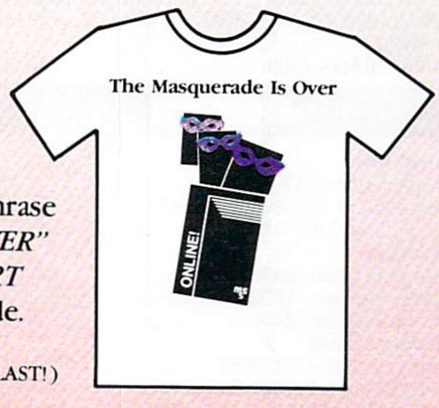
ONLINE! The Original Telecommunications Standard

Online! — a *legend* in its own time. All of us know that. Now we present Online! 2.0 to you... The Re-birth of a standard.

Online! 2.0 is not a mere face lift. We spent long Research and Development hours segregating *YOUR* letters and not only added valuable suggestions, but threw in a few trade secrets of our own.

So, if things like 9600 Baud without delay, KERMIT protocol, auto-chop and clip board support appeal to you, feel free to call us, or your local dealer with more inquiries. If the above specifications don't do a thing for you, perhaps an Online! telephone book to make calling multiple bulletin boards easier, a Review Buffer to re-read text that scrolled too fast at a later time and 132 column support will!

Now, take a second to think about who's masquerading, but only a second, and then concentrate on the *REAL* Online!



Call us now, repeat the phrase *"THE MASQUERADE IS OVER"* and receive a *FREE T-SHIRT* with your order or upgrade.

(OFFER GOOD WHILE SUPPLIES LAST!)

7 Years of Quality Software and Still Growing Strong!

SUGGESTED RETAIL $69.95

FOR DEALER NEAREST YOU: (800) 327-8724

MICRO-SYSTEMS SOFTWARE, INC.

The Amiga Connection

AMIGA COMPUTERS

New Amiga systems are being introduced. Call for pricing on Amiga 500 and Amiga 200 system packages.

Genlock Interface....249.00
256K RAM expansion....149.00

ACCESSORIES

DATA SHIELD
300 Watt Backup....379.00
500 Watt Backup....589.00
Turbo 350 Watt Backup....449.00
P150 Power Director w/Modem....119.00
S85 Surge Protector....69.99
FIFTH GENERATION
Logical Connection 256K....299.00
KALAMAR DESIGNS
3½" Disk Cabinet - Teak....14.99
5¼" Disk Cabinet - Teak....19.99
KENSINGTON MICROWARE
Master Piece....99.99
Printer stand....19.99

MONITORS

AMDEK
Color 600 Hi-Res (640x240)....399.00
MAGNAVOX
515 RGB/Composite....299.00
643 14" RGB....419.00
NEC
JC 1401 Multisync/RGB....In Stock
ZENITH
ZVM 1220/1230....(ea.) $99.99
ZVM 1330 16 color RGB....459.00
AMIGA
1080 Hi-Res Color....269.00

MODEMS

ANCHOR
Volksmodem 300/1200....139.00
Signalman Express 1200 EXT....199.00
Lightning 2400 Baud EXT....319.00
HAYES
Smartmodem 300....139.00
Smartmodem 1200....389.00
Smartmodem 2400....599.00
COMMODORE
Amiga 1680-1200 BPS....179.00
PRACTICAL PERIPHERALS
1200 BPS External....159.00

PRINTERS

JUKI 5510C PRINTER $379.00

EPSON
LX-86 120 cps....239.00
FX-85, FX-286....Call
EX-800, EX-1000....Call
LQ-800, LQ-1000, LQ-2500....Call
HI80 4 Pen Plotter....359.00
Free book w/purchase
HEWLETT PACKARD
Think Jet....399.00
JUKI
5510C Color Dot Matrix....Call
6000 12 cps Daisywheel....Call
6100 18 cps Daisywheel....Call
6200 30 cps Wide Carriage....Call
6300 40 cps Wide Carriage....Call
NEC
CP660 Color Printer....589.00
OKIDATA
Okimate 20 Color Printer....129.00
ML-182 120 cps Dot Matrix....219.00
ML-192 160 cps Dot Matrix....339.00
ML-193+, ML-292+, ML-293+....Call
STAR MICRONICS
NL-10 Font Compatible....Call
NX-10 120 cps Dot Matrix....219.00
TOSHIBA
P321 24 Wire 80 column....479.00
P341 24 Wire 136 column....589.00
P351 24 Wire 136 column....1049.00

DISK DRIVES

COMMODORE
Amiga 1010 3½"....219.00
Amiga 1020 5¼"....189.00
MICROBOTICS
20 mb Hard Drive....1299.00

DISKETTES

MAXELL
3½" DS/DD (10)....21.49
5¼" DS/DD (10)....10.99
SONY
3½" DS/DD (5 w/case)....9.99
3½" DS/DD (10)....20.49
5¼" DS/DD (10)....9.99
FLIP 'N FILE
25 Disk Tub 3½"....9.99
GENERIC
3½" DS/DD (10)....19.99
3½" DS/DD Bulk 50 Pack....89.99

SOFTWARE

Micro Systems
ORGANIZE $64.99

ACTIVISION
Borrowed Time....32.99
Hacker....32.99
Mind Shadow....32.99
AEGIS DEVELOPMENT
Animation/Images....79.00
BATTERIES INCLUDED
Isgur Portfolio System....169.00
COMMODORE
Textcraft w/Graphic Craft....59.99
Musicraft....79.99
TLC Logo....79.99
Amiga Pascal....79.99
Lattice "C"....119.00
Assembler....79.99
Lisp....156.00
Enhancer DOS 1.2....14.99
DISCOVERY SOFTWARE
Marauder Back-up....32.99
ELECTRONIC ARTS
Deluxe Paint....69.99
Archon....31.99
One on One....31.99
Sky Fox....31.99
Financial Cookbook....37.99
Seven Cities of Gold....31.99
Arctic Fox....31.99
Deluxe Print....74.99
Instant Music....34.99
Deluxe Video....69.99
INFOCOM
Wishbringer....31.99
Hitchhiker's Guide....31.99
Spellbreaker....31.99
Planetfall....31.99
Witness....31.99
MEGASOFT LTD
A-Copier....34.99
A-Report....44.99
MICRO ILLUSIONS
Dynamic-Cad....349.00
MINDSCAPE
Halley Project....31.99
Deja Vu....34.99
Keyboard Cadet....29.99
MICRO SYSTEMS
Analyze Version 2.0....119.00
Scribble....64.99
On-Line/Comm....49.99
SUBLOGIC
Flight Simulator II....37.99
V.I.P.
V.I.P. Professional....139.00

CMO COMPUTER MAIL ORDER

In the U.S.A. and Canada
Call toll-free: 1 800 233-8950
Outside the U.S.A. 717 327-9575 Telex 5106017898
Corporate and Educational Institutions call toll-free: 1 800 221-4283
C.M.O. 477 East Third Street Dept. B905, Williamsport, PA 17701
All major credit cards accepted

POLICY: Add 3%, minimum $7.00 shipping and handling. Larger shipments may require additional charges. Personal & company checks require 3 weeks clearance. For faster delivery use your credit card or send cashier's check or bank money order. PA residents add 6% sales tax. Defective software will be replaced with same item only. All items subject to availability and price change. All sales final, returned shipments are subject to restocking fee.

Circle 316 on Reader Service card.

DH

BASIC Intuition

Use custom screens and windows to create professional-looking Amiga Basic displays, and discover how event trapping lets you keep your programs on track.

By Bryan Catley

A central challenge for any programmer is to create programs that allow easy input and easy viewing of output. On the Amiga, the routines in the Intuition library let C and assembly-language programmers construct easily accessible programs. In this article I will show you how you can write BASIC programs that rival commercial software in their ease of use, by using the built-in feaures of Amiga Basic, and by applying event-trapping techniques.

Windows and Screens

Amiga programs communicate with users via windows. For instance, on the right of the Amiga Basic display is a half-sized window named List, which is used to enter new programs and to edit existing ones. Behind the List window is the Output window where you enter immediate commands and where Amiga Basic displays program output. All programs that require input from the user, and that have to display results, need a window.

All Amiga windows are associated with a screen that defines their environment. Screens determine the maximum resolution and the number of colors available to associated windows. Since Amiga Basic uses the Workbench screen, its List and Output windows default to the resolution (640 × 200 pixels) and colors (four) of that screen. However, the Workbench parameters will not suit your needs all the time, so you can define your own environment by typing SCREEN *number, width, height, depth, mode* into your program.

The first parameter of the Screen command is an identifying number from one to four, used by other commands to reference the screen. Width and height, measured in pixels, define the screen size. Width must be 640 for a hi-res screen, or 320 for low-res. The height must be 200 for a non-interlaced screen, or 400 for interlaced.

Depth specifies the number of bit-planes that are to be used in determining the color of each pixel on the screen. A bit-plane consists of an area of memory where each bit represents one pixel on the screen, anywhere from 8,000 bytes (64,000 bits) for a 320 × 200 screen to 32,000 bytes (256,000 bits) for a 640 × 400 screen.

Since any one bit can have only the values 0 or 1, a screen with one bit-plane can only display two colors. However, if you add a second bit-plane, each pixel is now represented by two bits, and you have four possible values representing four different colors. So, each time you add another bit-plane, you double the number of possible colors available on that screen. Each screen has its own set of bit-planes. Hi-res screens are restricted to a maximum of four bit-planes with 16 colors, while low-res screens may have up to five bit-planes with 32 colors.

Mode is a reference number that identifies a screen's resolution—one for 320 × 200 non-interlaced, two for 640 × 200 non-interlaced, three for 320 × 400 interlaced, and four for 640 × 400 interlaced. A non-interlaced screen has a vertical resolution of 200 pixels, which results in 200 horizontal lines stacked on top of one another.

These 200 lines are redrawn 60 times each second on your monitor, resulting in a steady picture. But with interlacing, each time the screen is drawn with 200 lines, things shift down half a line, and a second set of 200 lines is drawn. This effectively provides 400 lines on the screen, but the price is a distinct flicker since the entire screen is now only being drawn 30 times a second. The color combinations you choose also affect the amount of flicker you see.

An Open Window

Having defined a custom screen, your program opens a window on screen with the Window command: WINDOW *number, title, size, attribute, screen#*. Again, *number* is an identifier of one or above. But be careful, one is reserved for the Amiga Basic Output window. Title is ▶

ILLUSTRATED BY DAPHNE HEWETT

an optional character string displayed in the window's title bar. Size specifies the pixel coordinates of the upper left-hand and lower right-hand corners of the window, in the form (X,Y)–(X,Y). If it is omitted, the window will be full size.

Attribute is a reference number describing the window's gadgets. Zero means no gadgets or window refresh. One includes sizing gadgets, two a drag bar, four front/back gadgets and eight the close gadget. Requiring sizable memory, 16 refreshes a window after it has been covered or resized. Various combinations may be specified by adding the appropriate values; 10 provides a drag bar and close gadgets. The command defaults to 31. Screen# refers to the indentification number of the screen associated with the window.

When a window is opened, it is drawn on top of other windows already present on the screen, and becomes the current window. All future output will be directed to this new window. However, you may specify a different current window by issuing the command WINDOW *number*, where number identifies the new window. The referenced window will move to the front. If you wish to make a window current without moving it to the front, use the WINDOW OUTPUT *number* command.

Active and current windows are not necessarily the same. An active window is the window that receives keyboard input. You make a window active by clicking in it; note how the title bar changes. When you are finished with a window or a screen, be sure to close it with WINDOW CLOSE *number* or SCREEN CLOSE *number* to free allocated memory for other purposes.

Setting up your own custom menus in Amiga Basic means defining them, turning them on, and waiting for the user's selection. You define each menu, and the items within it, with a Menu statement.

Custom Menus

Setting up your own custom menus in Amiga Basic means defining them, turning them on, and waiting for the user's selection. You define each menu, and the items within it, with a Menu statement: MENU *menu-number, item-number, status, title.* Menu-number identifies the menu, the left-most menu being number one, the next number two, and so on. Item-number identifies the item within the menu. Zero specifies the menu header, one is the first item, two is the second, etc. The status numbers, zero to two, respectively disable a menu item, enable it or enable it with a check mark. The menu's title may be omitted from a previously defined item to change the status. If set to null (""), it turns off that menu. Grouped together, these statements define a complete set of menus. You put them in the program before they are actually required.

You activate menus with the MENU ON command. At this point, however, you must decide whether you want to poll the function MENU(0), or use event trapping to wait for the user to select a menu item. In polling, the function MENU(0) returns a 0 until a menu item is selected, then returns the number of the selected menu. To set up a polling loop type: M = 0:WHILE M = 0:M = MENU(0):WEND.

When execution falls through to the next statement, the user has selected a menu item, and M contains the menu number. It then becomes a simple matter of using an ON M GOTO/GOSUB statement to branch to the routine that handles that menu. Once in the menu routine, you can use the function MENU(1) to determine which item was selected, for example I = MENU(1). As before, you can use this value as the object of an ON. . .GOTO/GOSUB statement to reach the desired item routine. When finished, you would probably return to the polling loop described above.

Event Trapping

The event trapping approach is more involved than polling, but it provides almost limitless flexibility. Either before or immediately following the MENU ON statement, type ON MENU GOSUB *label.* The label names a subroutine that will automatically be branched to every time the user selects a menu item, regardless of what else may be going on. The subroutine provides all the necessary processing, and must end with a RETURN statement.

Once the ON MENU statement has been executed, Basic checks to see if a menu item has been selected after the execution of every statement. If so, the program branches to the menu-handling subroutine. Upon completion of that routine, the RETURN takes you back to the statement following the one that was executing when the menu item was originally selected. Using this approach, the program can continue processing or it can wait for a menu item to be selected. Hence, you trap an event—the user choosing a menu item.

If your program doesn't have to do processing while waiting for the user to select a menu, you can put the program to sleep with the statements: X = 1:WHILE X = 1:SLEEP:WEND. The result is similar to the polling technique, but SLEEP is a special statement designed specifically to work with the ON MENU GOSUB *label.* Whereas the polling method is continually using CPU cycles while waiting, SLEEP conserves system resources. The program shuts itself down until a menu item is selected, then it comes back to life. Once the RETURN is issued, the program will go back to "sleep," unless the variable X has been changed. In this case, control will fall through to the next statement. In the multitasking environment of the Amiga, this is the preferred approach since other tasks will have more resources available to them while the program waits for input.

At some point you'll want to return to the four standard Amiga Basic menus. Unless you issue the MENU OFF and MENU RESET statements, your custom menus will remain after the program has finished.

Mouse Handling

The right mouse button controls the menus, but Amiga Basic also provides methods of checking clicks made with the left button, and determining the screen coordinates of the pointer when the clicks are made. There is no set-up, but like menus, you may poll or use event trapping. The function MOUSE(0) returns a zero until the left button is pressed; at that point, MOUSE(1) returns the X-coordinate of the pointer, while MOUSE(2) returns the Y-coordinate.

The polling technique is as simple as: ►

Nimbus presents the first accounting program made exclusively for small businesses using the Amiga computer

Do business, not accounting

The easy new accounting program for managers of small businesses who *hate* to do accounting. With all the other record keeping programs, you have to learn accounting. With Nimbus, you just start.

It's available now—
If your dealer can't deliver it immediately, give us a call.

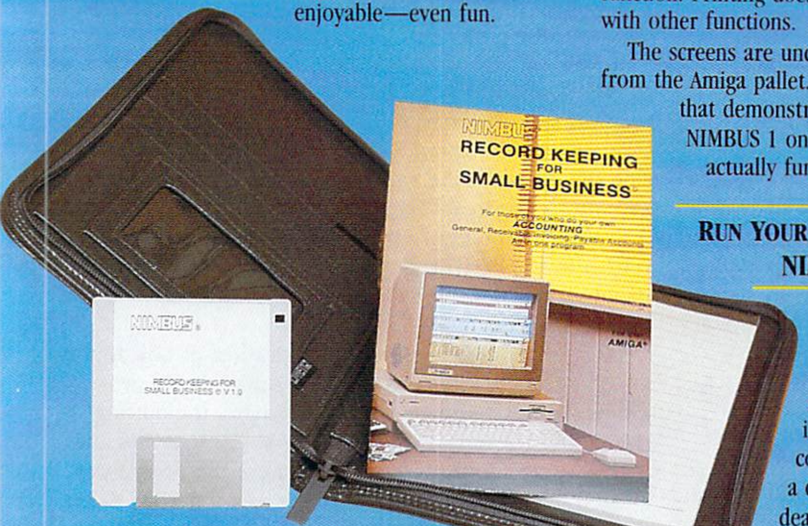
Simple Accounting Has Arrived

If you don't care how computer software works, only that it *does* work, this program is for you.

Nimbus 1 does the accounting, you run your business. The pain of accounting is gone.

What You Now Know About Accounting Is *Enough.*

No enormous manuals here—our instruction booklet is only 12 pages long. The difference is dramatically easier, more enjoyable—even fun.



At Last: Accounting Anyone Can Do

This is the first complete small business accounting software designed to take full advantage of the genius of the high speed Amiga computer.

This Is Instructionless Accounting

NIMBUS 1 requires no computer knowledge, no bookkeeping expertise. All of its functions are simple, self-explanatory, automatic—and *fast.* It's as easy as doing your checkbook. But don't confuse it with simple home budget programs.

NIMBUS 1: First, Fastest And The Most Fun

With NIMBUS 1, all the accounting functions are running concurrently. You just click on the mouse and move into a new function. Printing does not delay or interfere with other functions.

The screens are uncluttered, using colors from the Amiga pallet. Amiga dealers find that demonstrating accounting with NIMBUS 1 on the Amiga computer is actually fun.

Run Your Own Numbers On NIMBUS 1 Today

This is software you can use right now. Visit your Amiga dealer and try it. If NIMBUS 1 is not in stock, call us collect. We will rush a copy to you and your dealer.

Features Of The New Program Include:

- General, receivable and payable accounts are all interactive.
- Built-in ready to run chart of accounts and reports.
- Accounts kept on a cash basis. You know when customers pay you, not when they are supposed to.
- Equity and retained earnings accounts maintained automatically.
- Automatic computing of month-end and year-end closings.
- Prints invoices one at a time immediately, or later in a batch.
- Do three things at once. Each function has its own screen and you have instant access to each.
- All printing done in background, so you continue without interruption as your printer turns out reports.
- Daily, weekly, monthly and annual on-screen status reports.
- Programs and data require only one disk.
- Four color WYSIWYG input screens can be edited at any time. Make changes easily without a separate journal.
- No customer or vendor numbers to key in. One keystroke and one mouse-click gets any name from hundreds in less than half a second.
- Automatic pop-up mini-menus guide users through each function.
- Balance sheet, income statement, accounts receivable and accounts payable aging/status reports printed any time and automatically at end of month.
- Sold without copy protection for user convenience.

Call Nimbus for the name of the dealer nearest you. **(619) 756-5756**

Cost: only $149.50 complete with its executive zippered notebook

You can order by phone

NIMBUS

P.O. Box 1433
Rancho Santa Fe, California 92067
(619) 756-5756

Circle 324 on Reader Service card.

WHILE MOUSE(0) = 0:WEND :'Wait for click
X = MOUSE(1):Y = MOUSE(2) :'Pick up X and Y 'coordinates

Event trapping for the mouse is almost as easy:

MOUSE ON:ON MOUSE GOSUB label :'Set up event 'trap
N = 1:WHILE N = 1:SLEEP:WEND:'Go to sleep till click
MOUSE OFF:END:'Quit when N<>1

You can't trap keyboard events in Amiga Basic, you can only poll them. It is your responsibility to check for and handle special keys including the Return key. The polling may be performed with: X$ = "":WHILE X$ = "":X$ = INKEY$:WEND. The Speech Demonstration program that comes with Amiga Basic contains a good example of keyboard polling and mouse event trapping. Of course, you can always use the INPUT statement to control keyboard input.

Keep On Trappin'

Event trapping is not restricted to menus and the mouse. BREAK ON:ON BREAK GOSUB provides a means of receiving control when the user presses CTRL-C, and is used to terminate a program in midstream. When using animation, COLLISION ON:ON COLLISION GOSUB provides a method of detecting and handling collisions of two graphics elements or of a graphics element with a window border. With TIMER ON:ON TIMER(*n*) GOSUB the named routine is branched to automatically every *n* seconds. ON ERROR GOTO lets you specificy a routine to receive control when a fatal error occurs. The routine must terminate with a RESUME statement.

Events may be turned on and off with event ON and event OFF statements, but you can also issue event STOP statements. STOP temporarily turns off the trap for that event, but BASIC remembers if the event occurs, and the named routine is given control the next time event ON is issued.

Putting it all Together

The BAS.Intuition program exemplifies how to combine your own screens, windows and menus into one program. The program also contains routines that create an OK/Cancel requester and a string-input gadget from Amiga Basic. Studying it will show you how all the commands discussed in this article work together. The program may also serve as the basis for your own program by demonstrating how you use input from menus, the mouse, gadgets and requesters to channel the flow of control in a program.

The program is well-documented with comments, so I won't bore you with a line-by-line description. I will only point out a few important items. First, you, the programmer, should always know at what point the program is waiting for a mouse click or a menu selection, and you should be prepared to handle these events. You must also keep the user from performing actions that are not permitted at certain points. The event OFF statements are just as important as the event ON statements: Use them to restrict the user to valid options only.

Note also how event trapping works. By using the SLEEP command in WHILE loops, you are setting up a method that allows you to either stay in the loop or branch out. For instance, the only menu-handling routine that changes the value of X, and thereby causes the program to drop through to the exit routine, is the one that handles the Quit option. All other menu routines return to the WHILE loop with X unchanged. You decide when to return to a WHILE. . . SLEEP loop and when to fall through.

This program is a good demonstration of how to create an Intution interface for your Amiga Basic programs. You can improve it by improving the editing capabilities of the string gadget or by using BOBs and collision detection to determine where the mouse was clicked in requester gadgets. Use it as a template to create programs like the pros.□

Address all author correspondence to Bryan Catley, c/o AmigaWorld *Editorial, 80 Elm St., Peterborough, NH 03458*

***Listing 1.** BAS.Intuition.*

```
SCREEN 1,640,200,3,2         'medium-res screen with 8 colors.
WINDOW 2,"Template",,16,1    'creating a window on the screen.
MENU 1,0,1,"Project"         'name of the first menu.
MENU 1,1,1,"Quit"                'only option under the first menu
MENU 2,0,1,"Gadgets"         'name of the second menu.
MENU 2,1,1,"Print OK/Cancel"
MENU 2,2,1,"String Gadget"
MENU 3,0,0,""                'menus 3 AND 4 are null menus-use to
MENU 4,0,0,""                'cancel the BASIC Run and Windows menus.
ON MENU GOSUB MenuHandler 'identifies the menu-event trap routine.
MENU ON                       'activates the menus you defined.
x=1:WHILE x=1:SLEEP:WEND  'program waits for input until x changes.
Endit:                       'Before ending the program, close
  WINDOW CLOSE 2               'everything you opened and reactivate the
  SCREEN CLOSE 1               'BASIC menus. NOTE-Don't end a program
  MENU RESET                   'from a subroutine-end from main routine.
  END

MenuHandler:
  MENU OFF
  ON MENU(0) GOSUB Menu1, Menu2,  'which menu was chosen?
  MENU ON
  RETURN

Menu1:          'Menu 1 has only one option—the Quit option.
  x=0           'This deactivates the menu event trap and
  RETURN          'causes the program to execute the Endit routine.

Menu2:
  ON MENU(1) GOSUB M2item1, M2item2  'which item from menu2?
  RETURN

   M2item1:                      'example OK/Cancel requester.
     WINDOW 3, "Program Request",(200,90)-(500,150),0,1
     LOCATE 2,1
     PRINT "Make sure your printer is on-line"   'requester message.
     GOSUB GadgetMake
     ON MOUSE GOSUB OKHandler 'identify mouse-handling routine.
     MOUSE ON                          'turn on mouse trapping.
     z=1:WHILE z=1:SLEEP:WEND          'wait for a mouse event.
     WINDOW 2                     'make "Template" current window.
```

Listing continued on p. 40

Circle 323 on Reader Service card.

Release The Graphic Designer In You With PrintMaster Plus

A Partner in Design

Jump into the creative process anywhere you choose — graphics, format or copy. PrintMaster Plus is exceptionally flexible. And with over 100 pieces of clip art, 10 type fonts in 5 styles, and 11 border designs to select from, you'll have an endless variety of creative choices to make.

Time and time again, you've wanted to create exciting banners, posters, greeting cards, stationery, or calendars. If only you had the time *and* the budget to complete these projects yourself, you would!

Well, now you can. Thanks to the creative power in *PrintMaster Plus* — the innovative new graphic design software from Unison World.

In just a matter of minutes, you'll be able to see your great graphic ideas in print!

Best Of All . . . The Price!

PrintMaster Plus is *very* affordable graphic design software. So now you *do* have the time *and* the budget to develop the undiscovered graphic designer in you!

To get your copy of PrintMaster Plus, just ask your local computer dealer.

Available on MS-DOS compatible computers: Commodore Amiga, Atari 520 ST, Commodore 64/128, and Apple II computers.

Unison World
2150 Shattuck Avenue, Suite 902
Berkeley, California 94704
(415) 848-6666

Unison World is a division of Kyocera Unison, Inc.

```
    WINDOW CLOSE 3                          'kill the requester.
    IF p=1 THEN PRINT"Printing Activated"  'actions taken as result
    IF p=0 THEN PRINT"Printing Canceled" 'of input from requester.
    RETURN

  M2item2:                        'example of string requester.
    WINDOW 3, "String Gadget",(100,50)-(400,100),0,1  'gadget window.
    LOCATE 2,1
    PRINT "Enter Filename"
    LINE  (20,15)-(280,25),,b
    LOCATE 3,4:c=0:file$="":x$="":WINDOW 3
    WHILE x$<>CHR$(13) AND c < 15
     x$=INPUT$(1)
     IF x$> CHR$(32) THEN
       PRINT x$;
       file$=file$+x$
       c=c+1
     END IF
   IF x$= CHR$(8) AND c>0 THEN
     file$=LEFT$(file$,c-1)
     PRINT CHR$(8);
     c=c-1
   END IF
  WEND
  GOSUB GadgetMake
  ON MOUSE GOSUB OKHandler
  MOUSE ON
  z=1:WHILE z=1: SLEEP:WEND
  WINDOW 2
  WINDOW CLOSE 3
  IF p=1 THEN PRINT "Loading file: ";file$
  IF p=0 THEN PRINT "Aborting file: ";file$
  WINDOW 2
  WINDOW CLOSE 3
  RETURN

GadgetMake:
  LINE (50,30)-(120,45),3,b      'create action boxes.
  LINE (175,30)-(245,45),3,b
  LOCATE 5,11:PRINT "OK
  LOCATE 5,24:PRINT "Cancel"
  RETURN

OKHandler:
 MOUSE OFF
 IF WINDOW(0) <> 3 THEN WINDOW OUTPUT 3:MOUSE ON:RETURN
 'clicked in wrong window.

 m=MOUSE(0)   'we know the mouse was clicked-
              'this is FOR the NEXT line.
 A=MOUSE(3):b=MOUSE(4)      'mouse coordinates at
                           'MOUSE(0) FUNCTION call.
 IF (A > 50 AND A < 120) AND (b > 30 AND b < 45) THEN GOSUB OK:RETURN
 IF (A > 175 AND A < 245) AND (b > 30 AND b < 45) THEN GOSUB Cancel:RETURN
 MOUSE ON
 RETURN       'clicked in window but outside requester boxes.

  OK:                                 'clicked in OK box.
    LINE (50,30)-(120,45),3,bf       'inverse the box.
    p=1                              'flag indicates selected box.
    z=0                  ' turn off mouse trapping.
    RETURN

  Cancel:                             'clicked in cancel box.
    LINE (175,30)-(245,45),3,bf
    p=0                               'flag is different from above.
    z=0
    RETURN
```

The Machine To Unleash Your Imagination . . .

The Magazine To Explore Its Vision

For a computer as extraordinary as the Amiga,™ you need a magazine that can match its excellence, *AmigaWorld*.

AmigaWorld is the only magazine which provides you with ideas and information to get maximum performance from the Amiga's tremendous power and versatility.

Each issue gives you valuable insights to boost your productivity and enhance your creativity.

Whether you choose the Amiga as a serious business tool for its speed and multi-tasking capabilities . . . or for its superb graphics, drawing, color, (over 4,000 colors), and animation . . . or for its state-of-the-art music and speech . . . or for its scientific and CAD abilities, *AmigaWorld* can help you achieve superior results.

With its timely news features, product announcements and reviews, useful operating tips, and stunning graphics, *AmigaWorld* is as dynamic as the market it covers.

Don't wait! Become a charter subscriber and save 25% off the basic subscription rate and nearly 37% off the cover price. Return the coupon or the attached card. For immediate service, call toll free 1-800-258-5473. In NH, call, 1-924-9471.

☐ **Yes!**

I want to discover the full potential of this powerful machine. And save 25% off the basic rate as a charter subscriber. Enter my one year subscription (6 issues) to *AmigaWorld* for the low price of $14.97. If I'm not satisfied at any time, I will receive a full refund — no questions asked.

☐ Payment Enclosed ☐ Bill Me

Name ______________________

Address ______________________

City ______________________

State ____________ Zip ____________

Please make check payable to *AmigaWorld*. Canada and Mexico $17.97, Foreign Surface $34.97, Foreign Airmail $69.97. 1 year only, US funds drawn on US bank. Please allow 6-8 weeks for delivery.

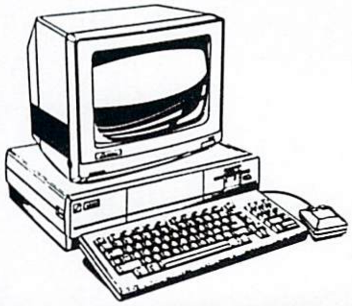
AmigaWorld
P.O. Box 868, Farmingdale, NY 11737

Amiga is a trademark of Commodore-Amiga, Inc.

37SB2

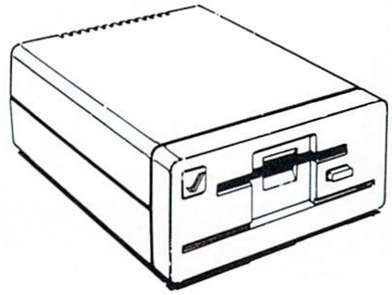
GO AMIGO!™

BRINGING THE WORLD OF AMIGA™ PRODUCTS TO YOU...FAST!



AMIGA 1000 COMPUTER
AMIGA 1080 COLOR MONITOR
(Call For Current Price)

AMIGA 1010 3.5″ DISK DRIVE
(Call For Current Price)



AMIGA MODEM 1200 RS
(Call For Current Price)

AMIGA 5.25″ DISK DRIVE WITH TRANSFORMER SOFTWARE — Available Now!
CALL FOR PRICE AND AVAILABILITY OF SIDECAR AND GENLOCK

XEBEC

20 MB SCSI Hard Disk

Fast Access
Easy To Use
Lots of Room!

$849

Outr-Aegis!

Get a
FREE Aegis T-Shirt
with any **TWO**
Aegis products.

AEGIS DEVELOPMENT

Diga!

AMIGA 500 & 2000	FUJI DS/DD Disks	OKIMATE 20 with Interface	SONY DS/DD Disks	Modem Special
	Box of 10 Disks		Box of 10 Disks	300/1200 Baud On-Line Software Cable
CALL	**$19.99**	**$199**	**$28**	**$199**

Tangible Graphics: Capturing Amiga Screens on Paper, Film and Videotape

Without doubt, the best medium for viewing Amiga graphics is an analog RGB monitor—such as the Amiga 1080 monitor. It's very inconvenient, however, to lug an Amiga system around with you every time you need to reproduce Amiga graphics. What you need is hardcopy—printouts, slides, or videotape—of what appears on your Amiga monitor. What follows are short, practical pieces on how to transfer Amiga images from your screen to paper, film and videotape.

Putting It on Paper

If you want to print Amiga screens, you must have a graphics-capable printer and a driver that supports it. The Amiga's system software has drivers for a number of graphics printers; you select the driver you need in Preferences.

The Amiga doesn't have a screen-dump program built into its system software. Instead, Commodore includes a utility program called *GraphicDump* with version 1.2 of Workbench. When you run GraphicDump by double-clicking on its icon, it pauses for 10 seconds to allow you to bring the screen you want to print to the front of the Workbench. Then, using the printer driver you selected from Preferences, GraphicDump prints out your Amiga screen.

Don't be fooled by the name GraphicDump: Everything, including text characters, will be printed by the program. Actually, this isn't surprising because everything the Amiga displays is bit-mapped graphics. Unlike many other computers, the Amiga doesn't have a separate text mode.

You can, to some degree, control the results of a GraphicDump printout by playing with the graphics printer settings in Preferences. You can use the horizontal and vertical settings, the margin and page-length settings to alter the size of your printouts, and, if you choose the black-and-white option, you can control which colors print as black and which print as white by changing the Threshold setting. (See Morton Kevelson's Graphic Hardcopy and the Amiga, *AmigaWorld*, March/April '87, page 36, for details on controlling printed output.) Experimentation will reveal what settings work best for you.

Bring 'Em Back Alive

What screen dump programs like GraphicDump can't do is allow you to modify the screens you print. To modify screens before printing them, you first have to capture them to disk as Amiga picture files. You can then load these pictures into an Amiga paint program, such as DeluxePaint II or Graphicraft, and modify them as you would any Amiga picture. You can then print the modified picture with the print routines built into the paint programs themselves.

Grabbit from Discovery Software and **Window Print II** from Computer Toolsmith are two commercial pro-

grams that let you capture screens as well as print them. They let you save screens in the Amiga IFF picture format, allowing you to load the pictures into any Amiga paint program. Once you've captured a screen, you don't necessarily have to print it. You can use a paint program (or the public-domain program See_ILBM) to display it for a screen shot or for videotaping. If you intend to produce hardcopy images of your Amiga screens, you'll need to be able to capture the screens to disk. —***The Editors***

GraphicDump
Commodore-Amiga
Included with Amiga Software Enhancer Kit
(Kickstart and Workbench 1.2)

Grabbit
Discovery Software
262 South 15th Street
Suite 400
Philadelphia, PA 19102

Window Print II
Computer Toolsmith
1055 Crest View Road
Vista, CA 92083

Successful Screenshots

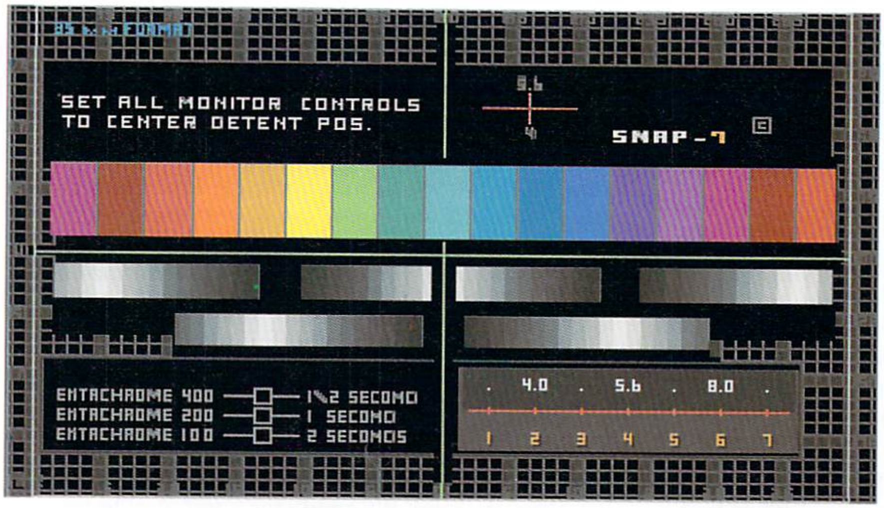


Photo 1. Typical color saturation test made with DeluxePaint. When making an exposure, use your mouse to point to the corresponding f-stop.

Reproducing colors accurately—getting the right color saturation—is one of the keys to taking successful screen shots of the Amiga screen. To control the color saturation of your slides and photographs, you need only take the time and effort to discover the optimal exposure settings for your camera-computer combination.

To control the color in your screen shots, you must perform a saturation test of your monitor screen. To perform the test, you first make a test target like the one in Photo 1. Note that the test pattern pictured here has a range of colors, a grey scale and boxes you use to indicate the f-stop of a particular shot. You can create such a target with any Amiga paint program.

Shooting Star

Once you have your target, set your camera up in front of the screen and darken the room as much as possible. Using a moderate-speed film like Ektachrome 200 and a shutter speed of one second, take a series of seven shots of the target using various mid-range f-stops on your camera. I take shots between f3.5 and f9. I find that these lens openings give me the sharpest picture.

When making an exposure, be sure that your mouse pointer is positioned over the corresponding f-stop on the target.

When your slides or photographs return from the processing lab, you will have a series of photographs in which the color saturation ranges from dark to bleached. If you positioned the mouse pointer correctly, the f-stop of each of the exposures should be evident in your pictures. Simply choose the picture that yields the best results and use the corresponding f-stop in all future screen shots. Note, however, that if you change anything about the exposure—film, shutter speed, monitor controls, processing lab, or distance between camera and screen—you will most likely change the optimal f-stop and have to perform another series of test shots.

Hints and Tips

I strongly recommend Ektachrome 200 film for screen shots; it sparkles and does the best job of nearly everything in the palette. I was surprised at how bad Kodachrome looked—it must be allergic to Amigas!

Forget your light meter unless it will average light off the screen. My high-tech digital light meter gave different results everytime I took a reading. A shutter speed of one second avoided the whole metering problem because the monitor is a consistent light source when averaged over that period of time. Better yet, hi-res flicker disappears with a one second exposure, although color saturation remains consistent at all resolutions.

If you're a knob tweaker like me, paste a note on the monitor controls door that says "For slides, set all controls in the detent position." Be certain that you take your screen shots from the same distance you shot your test pattern from to avoid inconsistent results. Pay attention to camera alignment—a crooked slide is as bad as a blurry one—and try to avoid putting straight lines near the edges of screens you want to shoot; such lines will appear curved on your photograph.

Remember, you can control the quality of screen shots on the Amiga; you just have to do a little testing. ▶

—***Paul Beck***

Imprint and Polaroid Palette

While you can produce acceptable screen shots with a 35mm or larger-format camera, better quality without curvature distortion is available if you want to invest the money. For the last three years, the **Polaroid Palette** Film Recorder has been one of the best devices for capturing images from microcomputer screens. However, it was only available to Apple II and IBM PC users.

Now, thanks to the company Liquid Light, and their **Imprint** system, the Amiga can be interfaced to the Polaroid Palette too. Imprint consists of a small hardware interface and some sophisticated (but "user-friendly") software that allows the Amiga to use the same Palette hardware developed for use with the IBM's CGA (four-color) graphic system, but allows access to all 4,096 colors the Amiga can produce.

The Polaroid Palette consists of a film recorder, a 35mm camera back, a Polaroid 3 ¼ × 4 ¼ instant camera back, and an instant slide processor and mounting device. You can use Imprint and the Palette to take full-color 35mm slides, (instant or processed), instant color pictures, color transparencies or even four-color separations of any picture (except Hold And Modify) the Amiga can generate. This includes 320 × 200, 320 × 400, 640 × 200 and 640 × 400 in one to five bitplanes of color.

An important aspect of the Imprint system is that interlaced images become perfectly suitable for image generation, regardless of the monitor you are using. The flicker associated with interlaced mode is not apparent when you use the Palette system.

Double Exposures

The Imprint software takes two different approaches to capturing the Amiga images. The main Imprint software works with disk-based pictures. You can use your favorite graphics design system to create the pictures, then save them to disk. When you are ready, you can load the Imprint software, choose the film type you are using, and load and expose the images. Or, you can choose another software option called Snapshot, which hides itself in the background. Then you can load and run a program, and with a simple keystroke, take a "snapshot" of what is on the screen. With these two methods, it becomes possible to capture virtually anything you place on the screen.

I have used Imprint with paintings and drawings made with DeluxePaint II, Images, Draw Plus, Impact and many other commercial programs. I have used it for slides of chemical formula for a technical lecture, slides of graphs and charts for a business presentation, and for making screen shots of software for various articles I have prepared.

If you are using the Amiga for the generation of professional computer graphics, you will no doubt find the Imprint system a godsend. Because regardless of how great your pictures are, if you can't use them in your work they are useless to you. Imprint goes a long way toward helping the Amiga fulfill its potential as a graphic design tool. ***—Lou Wallace***

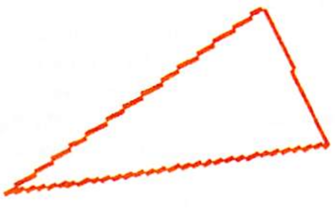
Imprint ($495.95)
Liquid Light Inc.
2301 West 205th Street
Suite 106
Torrance, CA 90501
213/618-0274
Requires a Polaroid Palette

Professional Images

If you need professional, reproduction-quality color prints, slides, negatives, and overheads of Amiga screens, contact **ImageSet Corp.** of San Francisco, CA. They produce images directly from IFF picture files and from Amiga Hold-And-Modify files. They can even provide you with color separations if you want to print full-color Amiga screens. Prices start at $9.30 for a 35mm color slide of an IFF picture and range up to $242 for an 8 × 10 color separation.

—The Editors

ImageSet Corp.
555 19th Street
San Francisco, CA 94107
415/626-8366



Available on Videotape

Capturing Amiga 1000 graphics and text on videotape is simple; you hook your VCR's Video-In port to the composite video port on the back of your computer with a double-ended RCA jack and press the Record button on the VCR. Whatever appears on your monitor will appear on videotape. . . sort of.

The problem with taping from the Amiga Video-Out port is that the quality of the Amiga NTSC video signal is not the best. Graphics and text that are crisp and colorful on an RGB monitor can bleed or appear washed out when transferred to videotape. You could get around this problem by purchasing the Genlock 1300 from Commodore, which offers a better Video-Out than the Amiga itself, or by using expensive video equipment to clean up the Amiga video signal. For home users, however, the inexpensive solution may be to work around the problem.

Happy Medium

As one of 150 affiliates of a national video-taping operation, I was attracted to the Amiga as a titling and graphics generator for my work. I quickly discovered the inadequacies of the Amiga 1000 video output, with its bleeding and bleached color combinations. Working further with DeluxePaint, I discovered that some foreground/background color combinations worked better than others. The results of my work with DPaint are summarized in the figure below, which charts the results of using 12 different foreground colors in text characters against 13 different backgrounds.

If you want to get the best possible results when creating titles for your home videos, avoid the unacceptable color combinations listed in the chart. Your videos will look better if you look carefully for suitable combinations. —*Bill Larned*

BACKGND → / FOREGND ↓	Yellow	Lt. Green	Med. Green	Dk. Green	Lt. Blue	Dk. Blue	Red	Amber	Dk. Brown	Purple	Gray	White	Black
Yellow		washed out	X	OK	X	X	OK	X	OK	OK	OK	dk. green shadow	X
Lt. Green	Darkens		OK	OK	OK	OK	X	OK	OK	X	lt. yellw shadow	OK	X
Dk. Green	X	OK	OK		OK	OK	X	light green	dk. grey shadow	X	OK	OK	light green
Lt. Blue	X	OK	OK	OK		OK	X	X	X	X	dk. blue shadow	dk. blue shadow	OK
Dk. Blue	X	OK	OK	OK	OK		X	lt. blue shadow	X	X	lt. blue shadow	lt. blue shadow	purple
Red	OK	orange shadow	X	X	X	X		X	orange/ red shdw	X	X	X	X
Amber	X	OK	X	X	X	X	X		dirty white	X	OK	OK	X
Dk. Brown	OK	amber shadow	X	X	X	X	X	OK		dirty white	X	light orange	X
Purple	X	X	X	X	X	X	X	OK	X			X	lighter
Gray	X	OK	X	whitish	X	whitish	X	OK	whitish	X		OK	whitish
White	X	OK	X	very white	X	very white	X	grey	OK	OK	OK		OK
Black	lt. green shadow	OK	OK	very black	OK	very black	X	OK	lt green shadow	grey shadows	OK	OK	

OK = acceptable foreground/ background combination

X = unacceptable foreground/ background combination

other notes = combinations that yield unexpected though usable results

MicroSearch Discoveries

"At MicroSearch, we listen to our customers . . . carefully"

City Desk Sets New Standards in Desktop Publishing

MicroSearch is about to set the standard for desktop publishing for the Amiga with its NEW City Desk Desktop publishing program. City Desk was designed, with *you* in mind, to be an integrated package from the start, with the emphasis on exploiting the versatility and simplicity of the Amiga. Written by SunRize Industries of College Station, Texas, (the developers of the digital sound sampler Perfect Sound, *see below*), the package will be available May 1, 1987. The retail price for City Desk will be $149.95.

Postscript compatability and kerning. Naturally, City Desk will have postscript compatability and kerning. Since these are rapidly becoming the standard in the desktop publishing industry and because you want them, City Desk is written to include these two features from the start.

- **Mix graphics and text on your page**
- **View and edit multiple pages**
- **Any number of columns per page**
- **Mouse moves and crops graphics and text**
- **Mix any number of fonts**
- **Use Laser printer for typeset look**
- **Draw lines and boxes**
- **Clip art included**

Integrate Graphics. With the graphics integration feature, City Desk is designed with the sales and marketing professional in mind as well as the graphics art industry. City Desk is ideal for making catalog updates and announcing price specials without messy cut and paste or the expense of typesetting. City Desk can be used by the graphics art professional for high quality page layout, highlighting easily with lines and boxes you draw yourself.

IFF file transfer for photographs. Integrating graphics is made possible by use of the industry standard IFF file format in the program. This allows you to use any file in IFF brush format, even digitized photographs or the included library of clip art, to get your message across.

We know what your customers want to buy. Use City Desk to make newsletters similar to this one and satisfy your customers. Dealers, your inquiries are invited. (Photograph has been retouched).

After loading the file, City Desk allows you to place the text or graphics *anywhere* on the page and easily enlarge, shrink, or crop the image or text by using the mouse.

Use other text and multiple fonts. City Desk will also allow you to load text from any of the Amiga word processors that are currently on the market. By using City Desk's powerful imbedded command codes and the Amiga's standard fonts, you can mix any number of different fonts to enhance your document! This lets you have *unprecedented* control of text fonts, sizes and styles.

Multiple pages and columns. With City Desk you can view multiple pages at one time and easily drag text and graphics from one page to another, as well as format any of those pages into as many columns as you feel are necessary. City Desk also lets you justify the text — right, left, or center — while it is in the column.

Laser printer compatability. City Desk lets you use the HP LaserJet+ for crisp typeset quality, or any other Preferences printer.

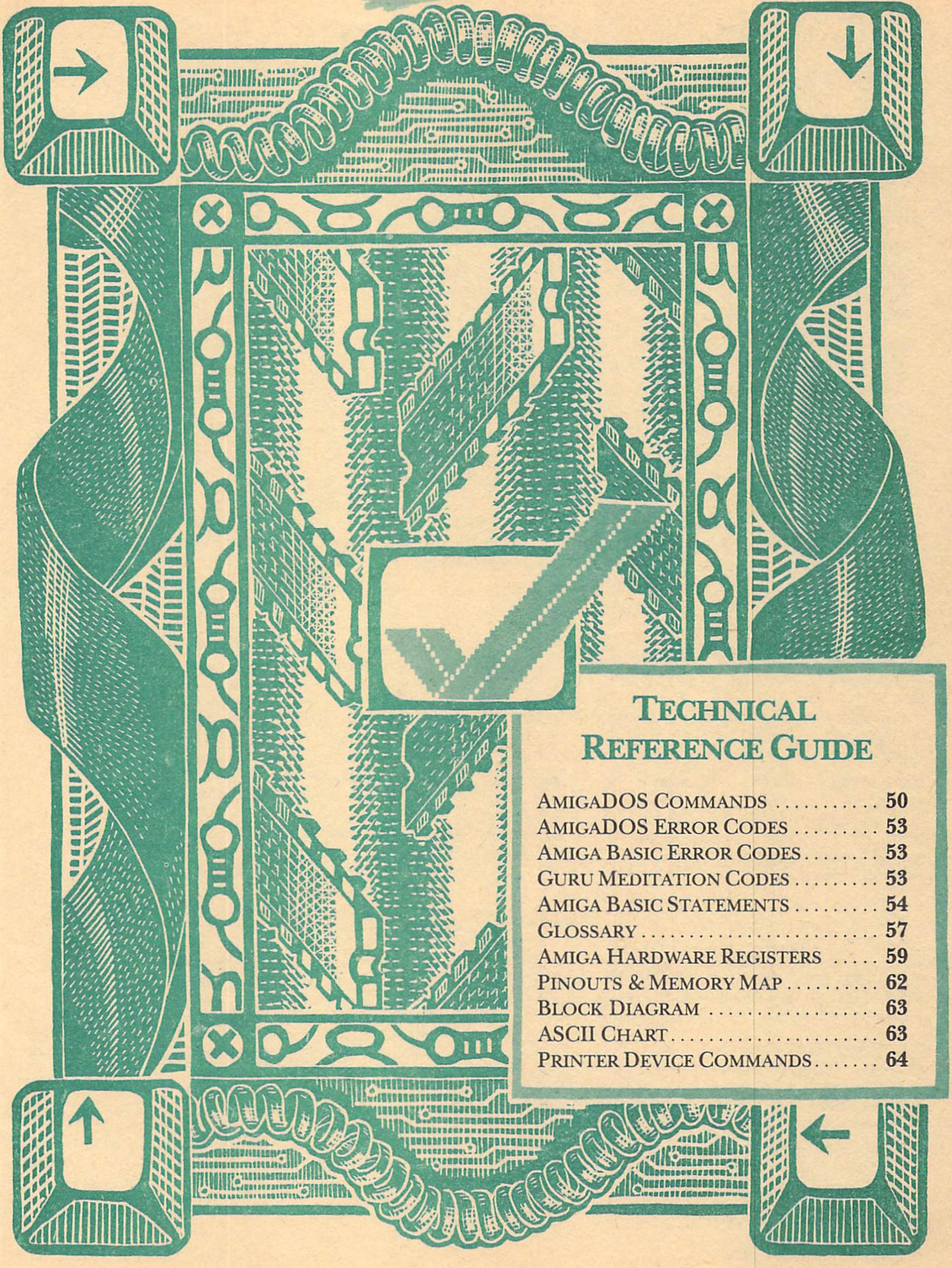
Perfect Sound Records in Stereo

MicroSearch has the Perfect Solution for stereo recording of digitized sounds. Perfect Sound! Other digital sound samplers can record only one channel at a time. Perfect Sound can record in true stereo simultaneously. Because of the popularity of this feature, there are over five thousand Perfect Sounds in use today.

SunRize Industries came up with the idea and they were able to successfully produce the finished product. Now you can flip, graph, insert or delete recorded sounds. Other capabilities allow you to create IFF instruments and change playback or record speeds. Perfect Sound comes with the "C" source code and a library of recorded sounds. Perfect Sound retails for $89.95.

MicroSearch, Inc.
9896 Southwest Freeway
Houston, Texas 77074
(713) 988-2818

Circle 313 on Reader Service card.



Technical Reference Guide

ILLUSTRATED BY BETH KROMMES

AMIGADOS 1.2 CLI COMMANDS

By William B. Catchings and Mark L. Van Name

Syntax Conventions and Explanatory Notes

- *Command names and keywords* are shown in upper case.
- *Arguments* that you must supply are in italic.
- *Optional arguments* are enclosed in square brackets, e.g., [*noicons*].
- If there are *several possible options* at any point in a command, these are presented within braces ({ }) and separated by vertical bars, e.g., [*a* | *b* | *c*]. If a command allows an argument to *repeat* several times, one argument will be shown, and it will be followed by an ellipsis, e.g., ***arg***. . . . These syntax symbols can be intermixed to express different alternatives; for example, the line:

COMMAND ***a*** { ***b*** | [**KEYWORD**] ***c*** }

says that **COMMAND** has one mandatory argument, a. It has one additional, optional argument, either b or c. If you choose *c*, you may optionally precede it with the word **KEYWORD**.

Most AmigaDOS commands can accept their arguments in one of two ways. The first is by position, i.e., you must supply the arguments in the order shown. In the syntax for each command given above, the parameters are shown in the order in which you must specify them if you are using positional notation. The other alternative is to specify parameters by keyword. If you do so, you must precede each parameter with its keyword, but the parameters can be in any order.

Many commands allow you to mix these two styles. For example, you can give the second parameter to the COPY command first, then follow it with the appropriate keyword and first parameter.

Note that some keywords are shown as optional, while others are not. Those that are not optional must be supplied even when you are giving arguments by position.

Nearly all AmigaDOS CLI commands will accept the single argument, ?. Given this argument, a command will display its syntax and then await input.

File and Directory Management

CD [***directory***]
Sets the current directory and/or drive. If you give no parameters, it displays the name of the current directory. If you are in a subdirectory, a slash (/) will move you up one directory level.

COPY [[**FROM**] ***source file***] **TO** ***destination file*** [**ALL**] [**QUIET**]
Copies the contents of a file or directory to another place. You may copy all or some of the files of a directory to another directory, or you may just copy one file to another. The **ALL** option causes AmigaDOS to copy the contents of all subdirectories as well.

DELETE ***name*** [***name*** . . .] [**ALL**] [**Q** | **QUIET**]
Deletes the named files and/or directories. You cannot delete a directory if it contains any files. If you specify a directory name and **ALL**, AmigaDOS will delete the entire directory and all files and subdirectories in it and its subdirectories. You can list up to ten files and/or directories. Any name also may contain an AmigaDOS file name pattern.

DIR [***directory***] [**OPT** { **A** | **I** | **AI** }]
Provides a listing of the files in the specified directory, or in the current directory if no name is specified. The **A** option will list the files in all subdirectories. The **I** option allows you to browse the directory interactively, one entry at a time.

FILENOTE [**FILE**] ***file*** **COMMENT** ***string***
Associates a comment with a file. The comment cannot be longer than 80 characters.

LIST [[**DIR**] ***dir or file***] [**TO** ***file***]
[{ **P** | **PAT** } ***pattern***]
[**KEYS**] [**DATES**] [**NODATES**]
[**S** ***search string***] [**SINCE** ***date***][**UPTO** ***date***] [**QUICK**]
Provides information about the specified directory or file, or, if none is given, the current directory. The options allow you to control the files about which you receive information either with a name pattern or a search string. You can also control the dates after or before which you want to see files, as well as several other options.

MAKEDIR ***directory name***
Creates a new directory with the name given.

PROTECT [**FILE**] ***file*** [**FLAGS**] ***status***
Sets the protection status of the specified file. The status may be any combination of **r** (read), **w** (write), **d** (delete) or **e** (execute). AmigaDOS will not let you delete a protected file that does not have the **d** status. You can check the status from within programs where appropriate.

RENAME [**FROM**] ***old name*** { **TO** | **AS** } ***new name***
Gives a new name to a file or directory. The names both must be on the same disk.

SETDATE ***file date*** [***time***]
Sets the date of a file to the specified date and, optionally, time.

TYPE [**FROM**] ***file*** [[**TO**] ***destination*** **file**] [**OPT** { **N** | **H** }]
Displays the contents of the specified file. The **N** option will provide line numbers, while the **H** option will display the file's contents in hexadecimal format.

Working with the Contents of Files

ED [**FROM**] ***file*** [**SIZE** ***n***]
Starts the ED screen-oriented text editor on the specified file. This editor reads the file into memory, and assumes a

memory workspace of 40K. If your file is larger, you must supply an adequate workspace size with the **SIZE** parameter.

EDIT [**FROM**] ***file*** [[**TO**] ***destination file***]
[**WITH name**] [**VER name**] [**OPT options**]
Starts the EDIT line editor on the specified file. You can use this editor to process the file sequentially, either with commands you enter or with commands stored in the file specified after the **WITH** option. The output goes either to the specified destination file or to a temporary file. The other options control where the editor sends its messages and on what lines it works.

JOIN ***file file*** [***file*** . . .] **AS** ***destination file***
Concatenates from two to fifteen files, in the order given, to form a new file.

NOTEPAD [[**–Q**] ***file***]
Runs the Notepad utility. If you give a file name, the Notepad will start up on that file. The **–Q** option stops it from reading fonts from the disk.

SAY [**–M**| **–F**] [**–R**| **–N**] [**–S##**] [**–P##**] [**–** ***xfilename|quoted string***]
Causes the Amiga to speak the specified string or the contents of the given file. If you give neither, a window will appear and prompt you for a string. The options allow you to control the gender, speed, and pitch of the voice, as well as whether it should sound "natural" (**–N**) or more like a robot (**–R**).

SEARCH [[**FROM**] ***name or pattern***]
[**SEARCH**] ***search string*** [**ALL**]
Searches the specified file(s) or, if none, all of the files in the current directory, for the given search string. You can give an individual file name or an AmigaDOS file pattern. The **ALL** option forces AmigaDOS to search all of the files below the current directory.

SORT [**FROM**] ***source file***
[[**TO**] ***destination file***]
[**COLSTART** ***n***]
Does a simple sort of the specified file. This command performs a simple textual sort and assumes that the input file is a text file. It is relatively slow and cannot sort any file that will not fit into memory. The COLSTART option lets you specify the starting column for the sort, but it always sorts using the characters from there to the end of each line.

Device Management

ADDBUFFERS DF*i*: *n*
Adds n extra buffers of 512 bytes each to the system cache for the specified drive. This costs memory but can improve disk access time.

ASSIGN [***name*** [***directory***] | **LIST**]
Gives a logical name to a directory. This command is used most often to tell AmigaDOS to look in a different place (e.g., a directory in RAM:) for files (e.g., the CLI commands).

BINDDRIVERS
Causes the system to bind to it all of the device drivers in the Workbench Expansion drawer. This is part of the means by which you can add new devices to your Amiga. This command usually appears only in the :S/STARTUP-SEQUENCE file on the Workbench disk.

DISKCHANGE DF*n*:
Informs the Amiga that the disk has been changed in the drive specified. This command is not necessary for the normal Amiga 3.5″ drives. However, you must use it for other drives, such as the 5.25″ drive, that do not detect automatically when the disk in them has been changed.

DISKCOPY [**FROM**] **DFi: TO DFi:** [**NAME** ***name***]
Copies the contents of one disk to another. While they no longer must be of the same type (as was the case with 1.1), there must be enough room on the destination disk to hold the contents of the source. The optional **NAME** parameter lets you name the new disk. The destination disk need not be previously formatted.

DISKDOCTOR DF*n*:
Attempts to recover the data on a damaged disk.

FORMAT DRIVE DF*n*: NAME *name* [**NOICONS**]
Formats a new disk. The disk will be given the indicated name. Unless you specify the **NOICONS** keyword, AmigaDOS will place a trashcan icon and directory on the disk.

INFO
Displays a line of information about each disk unit currently active on the system. If there are any files in the RAM: disk, this command will display a line about it also.

INSTALL [**DRIVE**] **DFi:**
Makes the disk in the specified drive a bootable disk. The disk must have been formatted already.

MOUNT ***device***
Tells the system to start using the indicated device. That device must have an entry in the file :devs/Mountlist on the Workbench disk.

RELABEL [**DRIVE**] **DFi:** [**NAME**] ***name***
Gives the specified new name to the indicated disk.

Basic System Management

DATE [{ ***date*** | ***time*** | ***date time*** }] [{ **TO** | **VER** } ***file***]
Sets or displays the current system date and time. If you give no arguments, it displays both. You can set either or both components.

PATH [**SHOW** | ***directory*** [***directory*** . . .] | **ADD** ***directory***
[***directory*** . . .]]
Tells AmigaDOS the directories in which to look (i.e., the "search path") for programs and commands that you attempt to run. With no arguments, it returns to the default search path (your current directory and then the directory assigned to C:). The **SHOW** argument causes it to display the current search path. Without the keyword **ADD** the list of directories shown replaces the old search path. With that keyword, that list is appended to the previous search path.

SETMAP { ***map file*** | **USA** }
Tells the Amiga to use an alternate key mapping for keyboard control and character display. There are new key maps for Germany, France, Great Britain, and Italy. If you specify USA, the system returns to the key map it has stored in ROM.

VERSION
Displays the current version number of the Workbench and Kickstart software that you are using.

CLI and Process Management

BREAK ***task*** [**ALL**] [**C**] [**D**] [**E**] [**F**]
Sets the specified attention flags (CTRL C through F) in the indicated task. The ALL option sets all four attention flags. This command is equivalent to running in that task and typing the corresponding control key(s).

CHANGETASKPRI ***pri***
Changes the priority of the current task. The system does not validate this priority, so be sure to stay in the range from

−5 to +5. This is most useful for causing background tasks to run with little impact on your main activity.

ENDCLI
Stops the current CLI session.

NEWCLI [*window*]
Starts a new CLI process and makes it the currently selected process. If a window definition is given, the window for the new CLI uses it. If not, the new CLI's window will have a standard size and position.

PROMPT *prompt string*
Changes the prompt in the current CLI to be the given string. This string may contain the characters %N, which will be replaced with the process number of the CLI.

RUN *command* [+ *command* . . .]
Executes the command(s) given in the background. It creates a non-interactive CLI process and gives that process the specified command(s) as input.

STACK [*size*]
Sets the stack size to the indicated size. Subsequent commands will use the new size. If no size is given, it displays the current stack size. The size is in bytes.

STATUS [*process*] [**FULL**] [**TCB**] [**SEGS**]
[**CLI** | **ALL**]
Displays information about all current processes or, if a single process number is given, about just that process. The options let you decide whether to see information about only CLI processes or about ALL processes, and what information you will see about each.

Command Files

ECHO *string*
Displays the given string. This command is normally used in command files to inform the user of progress.

EXECUTE *command file* [*argument* . . .]
Executes the given command file. The commands in it are executed one at a time, just as if you had typed them. Any arguments given are passed to the command file, where they may be substituted for dummy arguments in the file's commands. There are many command file options, and they are explained briefly in the AmigaDOS User's Manual.

FAILAT *n*
Stops a command sequence if any program returns an error code greater than or equal to the given number. Commands indicate that they have failed by sending a positive return code. In general, the larger the code returned, the worse the failure. This command lets you stop a command sequence when an error past a certain severity level has occurred.

IF [**NOT**] [**WARN**] [**ERROR**] [**FAIL**]
{ *str1* **EQ** *str* | **EXISTS** *file name* } *command*
[**ELSE** *command*]
ENDIF
Allows conditional execution of commands in command files. You can test to see if two strings, either of which may be an argument, are equal, as well as whether a file exists. You can also test to see if a warning (return code >= 5), error (>=10), or failure (>=20) occurred. There is an optional **ELSE** clause, and **IF**s can be nested.

LAB [*string*]
Makes the given string a label in the command file. If there is no string, this is an unnamed label line. Control flow in the command file can jump to labels by using the **SKIP** command.

QUIT [*n*]
Stops the command file and sends the specified return code. If none is given, the default is zero (i.e., successful termination).

SKIP [*label*]
Jumps in the command file to the indicated label. If no label is given, control jumps to the next (in sequential line order) **LAB** line that contains no string.

WAIT [*n*] [**SEC** | **SECS** | **MIN** | **MINS** | **UNTIL** *time*]
Causes execution to pause the indicated amount of time or until the indicated time.

Error Management

FAULT *n* [*n* . . .]
Displays the AmigaDOS messages that correspond to the specified fault code(s). You may display up to ten messages.

WHY
Explains why the previous command failed. Usually, typing this immediately after a failed command will give you a more detailed explanation of the failure than you first received.

Developer Commands

ALINK [**FROM** | **ROOT**] *object file*
[*file* . . . | + *file* . . .]
[**TO** *output file*] [**WITH** *file*]
[{ **LIBRARY** | **LIB** } *file*]
[**MAP** *file*] [**XREF** *file*] [**WIDTH** *n*]
Links individual object files and libraries together into an executable program. The options allow you to control such things as the libraries used and where the cross reference information is stored.

ASSEM [**PROG** | **FROM**] *program* [**−O***object* **file**]
[**−V***message file*] [**−L***listing file*]
[**−E***equates file*]
[**−** **C***options* | **OPT** *options*]
[**−I***include directories* . . .]
Assembles a program written in MC68000 assembly language into object code. The options allow you to control such things as where the listing and object files go.

DOWNLOAD [**FROM**] *source file* [**TO**] *destination file*
Allows you to download data from another machine (originally, a Sun workstation) to your Amiga. To use it, you must have a billboard and the program binload on the source machine.

READ [**TO**] *file* [**SERIAL**]
Reads data from either the parallel (default) or the serial (with the keyword) port into the specified file. It expects pairs of hexadecimal characters, each of which is stored as a byte. Input must be terminated with a Q.

Miscellaneous Commands

;
Allows you to add a comment to a command line. The CLI will ignore all characters on the line after the semicolon. It may be the first character on the line, i.e., you can have comment lines.

<*file*
Redirects the input of a command or program so that it comes from the given source (a file or device).

> *file*
Redirects the output of a command or program so that it goes to the given destination (a file or device).

AmigaDOS Error Codes *From the AmigaDOS manual, 2nd Edition*

Code	Meaning	Code	Meaning	Code	Meaning
103	Insufficient free store	209	Packet request type unknown*	220	Comment too big
104	Task table full	210	Invalid stream component name	221	Disk full
120	Argument line invalid or too long	211	Invalid object lock*	222	File is protected from deletion
121	File is not an object module	212	Object not of required type	223	File is protected from writing
122	Invalid resident library during load	213	Disk not validated	224	File is protected from reading
202	Object in use	214	Disk write-protected	225	Not a DOS disk
203	Object already exists	215	Rename across devices attempted	226	No disk in drive
204	Directory not found	216	Directory not empty	232	No more entries in directory*
205	Object not found	218	Device not mounted		* = programmer error
206	Invalid window	219	Seek error*		

Amiga Basic Error Codes *From Amiga Basic manual, Appendix B*

Code	Meaning	Code	Meaning	Code	Meaning
1	NEXT without FOR	17	Can't CONTINUE	53	File not found
2	Syntax error	18	Undefined user function	54	Bad file mode
3	RETURN without GOSUB	19	No RESUME	55	File already open
4	Out of data	20	RESUME without error	57	Device I/O error
5	Illegal function call	22	Missing operand	58	File already exists
6	Overflow	23	Line buffer overflow	61	Disk full
7	Out of memory	26	FOR without NEXT	62	INPUT past end
8	Undefined label	29	WHILE without WEND	63	Bad record number
9	Subscript out of range	30	WEND without WHILE	64	Bad file name
10	Duplicate definition	35	Undefined subprogram	67	Too many opened files
11	Division by zero	36	Subprogram already in use	68	Device unavailable
12	Illegal direct	37	Argument count mismatch	70	Permission denied (disk write-protected)
13	Type mismatch	38	Undefined array	73	Advanced feature
14	Out of heap space	50	Field overflow	74	Unknown volume
15	String too long	51	Internal error	21, 39-49, 69, 71-73, 75, 76 78-255	
16	String formula too complex	52	Bad file number		Unprintable errors

Guru Meditations *From the file include/exec/alerts.h*

Format of Guru Meditation messages:

Subsystem ID	General Error	Specific Error	Address of Task
00	00	0000	00000000

Subsystem ID Codes

CPU		Libraries		Devices		Resource		Other	
CPU Trap*	00	Exec	01	Audio	10	CIA	20	BootStrap	30
		Graphics	02	Console	11	Disk	21	Workbench	31
		Layers	03	GamePort	12	Misc	22	DiskCopy	32
		Intuition	04	Keyboard	13				
		Math	05	TrackDisk	14				
		Clist	06	Timer	15				
		DOS	07						
		RAM	08						
		Icon	09						
		Expansion	0A						

Note: If the first digit of the subsystem ID is greater than 3, there is no way to recover from the error. In these cases, subtract 8 from the first digit to get the subsystem ID number.

General Error Codes -00 if not applicable

Insufficient memory	01
MakeLibrary error	02
OpenLibrary error	03
OpenDevice error	04
OpenResource error	05
I/O error	06
No signal	07

Specific Alert Codes

Exec Library

81000001	68000 exception vector checksum
81000002	ExecBase checksum
81000003	library checksum error
81000004	no memory to make library
81000005	corrupted memory list
81000006	no memory for interrupt servers
81000007	InitAPtr
81000008	semaphore corrupt
81000009	free twice
8100000A	bogus exception

Graphics Library

82010001	no memory for copper display list
82010002	no memory for copper instruction list
82000003	copper list overload
82000004	copper intermediate list overload
82010005	no memory for copper list head
82010006	long frame, no memory
82010007	short frame, no memory
82010008	no memory for flood fill
82010009	text, no memory for TmpRas
8201000A	no memory for BltBitMap
8201000B	region memory
82010030	MakeVPort
82011234	GfxNoLCM

Layers Library

83010001	LayersNoMem

Intuition Library

84000001	unknown gadget type
04000001	recoverable form of previous message
84010002	no memory to create port
84010003	item plane alloc, no memory
84010004	sub alloc, no memory
84010005	plane alloc, no memory
84000006	item box top less than RelZero
84010007	no memory to open screen
84010008	open screen, raster alloc, no memory
84000009	open sys screen, unknown type
8401000A	add SW gadgets, no memory
8401000B	no memory to open window
8400000C	Bad State Return entering Intuition
8400000D	Bad Message received by IDCMP
8400000E	Wierd echo causing incomprehension
8400000F	couldn't open the console device

DOS Library

07010001	no memory at startup
07000002	EndTask didn't
07000003	Qpkt failure
07000004	Unexpected packet received
07000005	Freevec failed
07000006	Disk block sequence error
07000007	Bitmap corrupt
07000008	Key already free
07000009	Invalid checksum
0700000A	Disk Error
0700000B	Key out of range
0700000C	Bad overlay

RAM Library

08000001	Bad segment list

Expansion Library

0A000001	Bad Expansion Free

TrackDisk Device

14000001	calibrate: seek error
14000002	delay: error on timer wait

Timer Device

15000001	bad request
15000002	bad supply

Disk Resource

21000001	get unit: already has disk
21000002	interrupt: no active unit

BootStrap

30000001	boot code returned an error

*CPU Traps (exceptions) are defined by the 68000 CPU and not by the Amiga system software. There are 256 vectors for CPU exceptions; the lower 64 are hardware defined, the upper 192 are user-definable interrupt vectors. Specific CPU traps you might see in a Guru are:

00000002	Bus error
00000003	Address error
00000004	Illegal instruction
00000005	Divide-by-zero
00000006	CHK instruction
00000007	TRAPV instruction
00000008	Privilege violation
00000009	Trace
0000000A	Op Code 1010
0000000B	Op Code 1111

Other CPU traps include autovectors for different level interrupts, TRAP instruction vectors, and reserved traps.

AMIGA BASIC STATEMENTS AND FUNCTIONS

Compiled by William B. Catchings and Mark L. Van Name

Syntax Conventions and Explanatory Notes

- *Statement and function names* and *keywords* are in capitals.
- *Arguments* that you must supply are enclosed in angle brackets, e.g., <x>.
- *Numbers* are indicated with single letters, such as <n> or <i>. If the number may be either a real or an integer quantity, we use a letter near the end of the alphabet; if it may be only an integer, we use <i> or subsequent letters.
- *Strings* are denoted with a letter and a dollar sign, e.g., <x$>.
- *Optional arguments* are enclosed in square brackets, e.g., [STEP].
- If *several possible options* are available at any point in a command, these are presented within braces and separated by vertical bars, e.g., { x | y | z }. Arguments that can *repeat* are shown in one of two ways: with an ellipsis (. . .) after the argument; or with the notation <arg-list>, which means that there can be one or more arguments, each separated from the previous one by a comma.

These symbols can be intermixed to express different alternatives. For example, the line:

```
FUNCTION <x> [ { <y> | [ KEYWORD ] <z> } ]
```

says that **FUNCTION** has one mandatory argument, <x>. It has one additional, optional argument, either <y> or <z>. If you choose <z>, you may optionally precede it with the word **KEYWORD**.

AmigaDOS Statements and Functions

```
CHDIR <string>
DATE$
FILES [ <directory or file name> ]
KILL <file>
NAME <old file name> AS <new file name>
TIME$
```

Control Flow Statements

```
FOR <variable> = <x> TO <y> [ STEP <z> ]
      <statement>. . . NEXT [ <variable-list> ]
GOTO <line>
IF <expression>
  { GOTO <line> [ ELSE <else clause> ] |
    THEN <then clause> [ ELSE <else clause> ] |
    THEN <statement block>
      ELSEIF <expression> THEN <statement block> . . .
      ELSE <statement block> END IF}
ON BREAK GOSUB { <label> | 0 }
ON COLLISION GOSUB { <label> | 0 }
ON ERROR GOTO <line>
ON <expression> GOSUB <line list>
ON <expression> GOTO <line list>
ON MENU GOSUB { <label> | 0 }
ON MOUSE GOSUB { <label> | 0 }
ON TIMER { ( <n> ) GOSUB { <label> | 0 } | GOSUB 0 }
WHILE <expression> [ <statement> . . . ]
         WEND
```

Data Conversion Functions

```
CDBL( <x> )
CHR$( <i> )
CINT( <x> )
CLNG( <numeric expression> )
CSNG( <x> )
FIX( <x> )
HEX$( <x> )
MKI$( <short integer expression> )
MKL$( <long integer expression> )
MKS$( <single precision expression> )
MKD$( <double precision expression> )
OCT$( <x> )
STR$( <x> )
VAL( <x$> )
```

Declarative Statements

```
COMMON <variable-list>
DATA <constant> . . .
DEFINT <letter range>
DEFLNG <letter range>
DEFSNG <letter range>
DEFDBL <letter range>
DEFSTR <letter range>
DIM [ SHARED ] <variable-list>
  [ ( <dimension-list> ) ]
OPTION BASE <i>
READ <variable list>
RESTORE [ <line> ]
SHARED <variable list>
```

Error Handling Statements

```
ERR
ERL
ERROR <integer expression>
```

Function Declaration and Management Statements

```
{ CALL <name> ( [ <argument> . . . ] ) |
<name> [ <argument> . . . ] }
CHAIN [ MERGE ] <file>
      [ , [ <expression> ] [ , [ ALL ]
            [ , DELETE <range> ] ] ]
DECLARE FUNCTION <function id>
         [ ( <parameter-list> ) ] LIBRARY
DEF FN <name> [ ( <parameter-list> ) ]
      = <function definition>
END SUB
EXIT SUB
GOSUB <line>
LIBRARY { <filename> | CLOSE }
RETURN [ <line> ]
SUB <subprogram name>
         [ ( <formal parameter-list> ) ]STATIC
```

Graphics and Intuition User Interface Functions

```
AREA [ STEP ] ( <x>, <y> )
AREAFILL [ <mode> ]
CIRCLE [ STEP ] ( <x>, <y> ) , <radius>
    [ , <color id> [ , <start> , <end>
          [ , <aspect> ] ] ]
CLS
COLLISION { ( <object id> ) } | ON | OFF | STOP }
COLOR [ <foreground color id> ]
    [ , <background color id> ]
CSRLIN
GET ( <x1> , <y1> ) - ( <x2> , <y2> )
  , <array name> [ ( <index list> ) ]
```

(see also GET under Input/Output Functions)

```
LINE [ [ STEP ] ( <x1> , <y1> ) ]
           - [ STEP ] ( <x2> , <y2> ) ,
           [ <color id> ] [ , B [ F ] ]
LOCATE [ <line> ] [ , <column> ]
MENU { <menu id> , <item id> , <state>
    [ , <title string> ] |
    RESET | ( 0 ) | ( 1 ) | ON | OFF | STOP }
MOUSE { ( <i> ) | ON | OFF | STOP }
OBJECT.AX <object id> , <value>
OBJECT.AY <object id> , <value>
OBJECT.CLIP ( <x1> , <y1> ) - ( <x2> , <y2> )
OBJECT.CLOSE [ <object id list> ]
OBJECT.HIT <object id> , [ <me mask> ]
        [ , <hit mask> ]
OBJECT.ON <object id list>
OBJECT.OFF <object id list>
OBJECT.PLANES <object id> , [ <plane> ]
          [ , <plane switch> ]
OBJECT.PRIORITY <object id> , <priority>
OBJECT.SHAPE <object id> ,
          { <definition> | <object2 id> }
OBJECT.START <object id list>
OBJECT.STOP <object id list>
OBJECT.VX { <object id> , <value> | ( <object id> ) }
OBJECT.VY { <object id> , <value> | ( <object id> ) }'
OBJECT.X { <object id> , <value> | ( <object id> ) }
OBJECT.Y { <object id> , <value> | ( <object id> ) }'
PAINT [ STEP ] ( <x> , <y> )
         [ , <paint color id> [ , <border color id> ] ]
PALETTE <color id> , <red> , <green> , <blue>
PATTERN [ <line pattern> ] [ , <area pattern> ]
POINT( <x> , <y> )
POS( <y> )
PRESET [ STEP ] ( <x> , <y> ) [ , <color id> ]
PSET [ STEP ] ( <x> , <y> ) [ , <color id> ]
PTAB( <x> )
PUT [ STEP ] ( <x> , <y> ) , <array>
```

```
    [ ( <index list> ) ] [ , <action verb> ]
```

(see also PUT under Input/Output Functions)

```
SCREEN { <screen id> , <width> , <height>
        , <depth> , <mode> |
        CLOSE <screen id> }
SCROLL ( <x1> , <y1> ) - ( <x2> , <y2> ) ,
    <pixels to scroll right> ,
    <pixels to scroll down>
STICK( <i> )
STRIG( <i> )
WINDOW { <window id>
                    [ , [ <title> ]
                     [ , [ (<x1> , <y1>)-(<x2> , <y2>) ]
                      [ , [ <type> ]
                       [ , <screen id> ] ] ] ] |
       CLOSE <window id> | OUTPUT <window id> |
       ( <i> ) }
```

Input/Output Functions (not graphics or audio)

```
CLOSE [ [ # ] <file number> ] . . .
EOF( <file number> )
FIELD [ # ] <file number> , <field width>
    AS <string variable> . . .
GET [ # ] <file number> [ , <record number> ]
```

(see also GET under Graphics Functions)

```
INKEY$
INPUT [ ; ] [ <prompt string> ; ] <variable list>
INPUT$ ( <x> [ , [ # ] <file number> ] )
INPUT# <file number> , <variable list>
LINE INPUT [ ; ] [ " <prompt string> " ; ]
        <string variable>
LINE INPUT# <file number> ; <string variable>
LOC( <file number> )
LOF( <file number> )
LPOS( <x> )
LPRINT { [ <expression list> ] |
        USING <string expression> ; <expression list>
LSET <string variable> = <string expression>
OPEN { <MODE> , [ # ] <file number> , <file>
    [ , <file buffer size> ] |
    <file> [ FOR <mode> ] AS [ # ] <file number>
    [ LEN = <file buffer size> }
PRINT { <expression list> |
    USING <string expression> ; <expression list>
PRINT# <file number> ,
        [ USING <string expression> ; ]
          <expression list>
PUT [ # ] <file number> [ , <record number> ]
```

(see also PUT under Graphics Functions)

```
RSET <string variable> = <string expression>
TAB( <i> )
WIDTH [ <output device> | #<file number> |
    LPRINT ] [ <size> ] [ , <print area> ]
WRITE [ <expression list> ]
WRITE# <file number> , <expression list>
```

Mathematical Functions

```
ABS( <x> )
ATN( <x> )
COS( <x> )
EXP( <x> )
INT( <x> )
LOG( <x> )
SGN( <x> )
SIN( <x> )
SQR( <x> )
TAN( <x> )
```

Memory Access Functions

```
PEEKL( <memory address> )
PEEK( <memory address> )
PEEKW( <memory address> )
POKE <i> , <j>
POKEL <memory address> , <value>
POKEWL <memory address> , <value>
SADD( <string expression> )
VARPTR( <variable name> )
```

Miscellaneous Statements

```
LBOUND( <array name> [ , <dimension> ] )
[ LET ] <variable> = <expression>
RANDOMIZE { [ <expression> ] | [ TIMER ] }
REM <comment>
RND [ ( <x> ) ]
SWAP <variable1> , <variable2>
UBOUND( <array name> [ , <dimension> ] )
```

Program Management and Control Statements

```
BREAK { ON | OFF | STOP }
CLEAR [ , <program memory> ] [ , <stack> ]
CONT
DELETE [ <line> ] [ - <line> ]
END
ERASE <array_variable-list>
FRE ( { -1 | -2 | <x> } )
LIST { [ <line> ] |
     [ <line> ] [ - [ <line> ] ] , " <file name> "
LLIST [ <line> ] [ - [ <line> ] ]
LOAD [ <file> [ , R ] ]
MERGE <file>
NEW
RESUME [ { 0 | NEXT | <line> } ]
RUN [ { <line> | <filename> [ , R ] } ]
SAVE [ <file> [ , { A | P | B } ] ]
SLEEP
STOP
SYSTEM
TIMER [ { ON | OFF | STOP } ]
TRON
TROFF
```

Sound Functions

```
BEEP
SAY " <string> " [ , <mode array> ]
SOUND { <frequency> , <duration>
        [ , [ <volume> ] [ , <voice> ] ] |
        WAIT | RESUME }
<string variable> = TRANSLATE$( " <string> " )
WAVE <voice> , <wave definition>
```

String Manipulation Functions

```
<x$> + <y$>
ASC( <x$> )
INSTR( [ <i> , ] <x$> , <y$> )
LEFT$( <x$> , <i> )
LEN( <x$> )
MID$( <string expression1> , <i> [ , <j> ] )
    = <string expression2>
MID$( <x$> , <i> [ , <j> ] )
RIGHT$( <x$> , <i> )
SPACE$( <x> )
SPC( <i> )
STRING$( <length> , { <ascii code> | <x$> } )
UCASE$( <string expression> )
```

GLOSSARY OF AMIGA TERMS

Compiled by William B. Catchings and Mark L. Van Name

Hardware

68000: The single-chip microprocessor designed by Motorola that is used in the Amiga, as well as in the Apple Macintosh and the Atari ST computer families.

Bus: The part of a computer's hardware that allows the different components to exchange data. On the Amiga the bus connects the CPU, the memory, the coprocessors, the disk drives and other devices. The Amiga 500 and 1000 have external bus expansion slots on the left and right sides, respectively.

Chip Memory: The first 512K of memory on the Amiga. This memory is accessible both by the Amiga's 68000 CPU and by its three coprocessor chips. The coprocessor chips can use only memory that falls within this range.

Coprocessor: A separate processor unit that assists the main CPU by doing a specific task. An example of this is a math coprocessor, such as the Motorola 68881, that handles complex mathematical computations. The Amiga includes a set of three coprocessors that assume many audio and video functions from the main CPU.

CPU: (Central Processing Unit) The part of a computer that decodes and executes the computer's basic instructions. (The 68000 is the Amiga's CPU.)

DMA: (Direct Memory Access) A means by which different chips in the Amiga system use the memory they require by writing or reading it directly, as opposed to working with that memory through normal CPU instructions.

Expansion Memory: RAM added to your system beyond its standard configuration.

Fast Memory: Expansion memory beyond the first 512K. It is referred to as "fast" because it is not accessible to the Amiga's coprocessors and therefore does not have to be shared by the CPU.

Kilobyte: 1,024 bytes (two to the tenth power). Often thought of as one thousand bytes and abbreviated by K, as in 256K (256 times 1024, or approximately 256,000 bytes).

Megabyte: Approximately one million bytes (1,024 times 1,024 bytes); often abbreviated by M or MB.

Modem: (*Mo*dulator/*Dem*odulator) A device that turns a computer's digital signals into a telephone's analog signals (modulation), and vice versa (demodulation). A pair of modems is necessary for two computers to commununicate over a telephone line.

Monitor: A video output device similar to a television set.

Mouse: A hand-controlled device that relates its movement to activities on the computer screen such as pointer control and program selections.

Mouse Port: An outlet to which a mouse is attached, which can also accommodate joysticks and other peripherals.

Parallel Port: One of the 25-pin outlets on the back of the Amiga used primarily for connecting with printers. "Parallel" indicates that all eight bits of a byte are sent simultaneously, on eight separate wires within the cable.

RAM: (Random Access Memory) Storage residing on a computer chip, any part of which can be read or written in a non-predefined, or random, order. (This differs, for example, from a cassette tape whose information is accessed in sequential order.) This memory is used by the computer for storing data and executing its instructions.

RJ11: The kind of connector (jack) used for modular telephones (and the Amiga 1000 keyboard).

ROM: (Read Only Memory) Computer storage, similar to RAM, except that it can only be read; the computer is not capable of writing to ROM memory. ROM is often used to store computer instructions that don't change (such as the Amiga 500's Kickstart code).

RS-232C: A standard that describes the signals used for asynchronous serial communications. An RS-232C cable would be used to hook the Amiga to a Modem or to a serial printer.

Serial Port: One of the 25-pin outlets on the back of the Amiga. This outlet is used for asynchronous serial communications. Serial indicates that the bytes are sent one bit after another. Asynchronous means that the bytes are sent at no specific time, as opposed to synchronous which sends bytes at pre-defined intervals. Asynchronous serial communications are commonly used for connecting computers over telephone lines and for many printers.

Software

AmigaDOS: The Amiga's disk operating system.

AutoConfig: The ability to make hardware beyond the standard configuration a part of the system *automatically*. All you have to do is connect the hardware; the operating system figures out the rest. This capability is provided by version 1.2 of AmigaDOS and is supported by most vendors' hardware upgrades, such as memory expansion boards.

Boot: To start a computer. It comes from the phrase "to pick yourself up by your own bootstraps." This is because the computer must first find a program to run to tell itself how to run any other programs.

CLI: (Command Line Interpreter) A program that allows the user to execute AmigaDOS commands using the keyboard, as opposed to the Workbench/mouse approach.

Drawers: Places on the Amiga Workbench where you can store programs or files. (Drawers are the visual counterparts to AmigaDOS directories.) Drawers open windows that display their contents.

Exec: System software composed of many routines that perform vital Amiga system functions. Exec routines control such services as multitasking, interrupt and input/output management.

Gadget: A graphics control device that you click on with the mouse to communicate with a program. Examples are window scroll gadgets, front/back, close and drag gadgets.

Icon: A graphic image that represents a computer object (disk, program or file) and with which you can in some ways manipulate that object.

IFF: (Interchange File Format) An Amiga software standard (developed primarily by Electronic Arts) that allows for the interchange of graphic and sound files among programs of different type and from different companies. IFF supports numeric data, dates, pictures, text and sound.

Intuition: The part of the Amiga's operating system that provides the graphic images that the user sees and uses to manipulate the basic system. Intuition

is a set of routines that can be used by any program running under AmigaDOS to display gadgets, menus, requestors and icons, and to receive input from the keyboard, mouse and other devices. By using these routines, different types of programs can have consistent user interfaces.

Kickstart: The part of the Amiga's operating system that is needed to boot the Amiga and run any program. On the Amiga 1000, Kickstart is loaded from disk into a protected area of RAM (called "RAM-ROM"); the Amiga 500 and 2000 models have Kickstart in hardware ROM.

Menu: A graphics "box" usually containing selections and usually accessed from the menu bar using the mouse.

Multitasking: The ability of a computer to do more than one task at a time; i.e., run more than one program at a time. Unlike the other popular micros, the Amiga's operating system was designed to do multitasking.

OS: (Operating System) The software program that is always running on a computer. This program manages the hardware devices such as the screen and the disk drive and allows other programs to run.

Pointer: An object, used for making selections, that moves on the screen as you move the mouse. The standard Amiga pointer is an arrow.

Projects: Amiga parlance for things (files) created by Tools (programs). Notepad notes and DeluxePaint pictures are Projects.

Requester: A rectangular gadget into which you must type information before continuing in a program.

Scroll Bar: Scroll bar gadgets allow you to see information in a window that is outside the current viewing area of the window.

Sizing Gadget: A gadget that allows you to resize a window.

Tools: An Amiga term for a program running on the Workbench. The Notepad, Icon Editor and Graphicraft are Tools.

Window: A framed area of the screen that can contain an independently-running program and is usually manipulatable with gadgets. The Amiga's multitasking OS allows it to display many windows simultaneously.

Workbench: The Amiga's icon-based user software. This part of the OS allows you to work with a single graphics interface to manipulate files, run programs and perform other Amiga functions. Workbench uses the Intuition user interface routines to supply this graphical interaction.

Audio

Audio Coprocessor Chip: Nicknamed *Paula*, this chip controls many of the Amiga's audio abilities. It handles the actual digital-to-analog sound conversion for up to four voices within the Amiga. This chip also handles many of the input/output tasks for the Amiga, including disk and interrupt control, the mouse/joystick ports and the serial port.

MIDI: (Musical Instrument Digital Interface) The electronics standard for connecting electronic instruments digitally. MIDI is used by synthesizers, rhythm machines and other electronic instruments. The Amiga's audio coprocessor has the ability to drive a MIDI interface. Though the Amiga does not come with built-in MIDI hardware, it can be purchased relatively inexpensively.

Voice: Relating to the Amiga: A discrete electronically-produced sound. A voice can be any tone or instrument sound, including percussion sounds. The Amiga's four-channel sound hardware can simultaneously produce four voices, two each on the left and right stereo channels.

Graphics/Video

Animation Coprocessor Chip: Nicknamed *Agnus*, this chip controls several important graphic functions of the Amiga. It contains the Blitter and the Copper (see below), and also controls DMA.

Bit Map: An area of memory that the Amiga treats as a rectangular array of pixels. This portion of memory acts as the basis of the Amiga's screen graphics.

Blitter: A part of the animation coprocessor chip that can quickly draw lines and manipulate rectangular parts of graphic images. The term blitter comes from "blt," for BLock Transfer, since it moves blocks of bit-mapped data.

BOB: (Blitter OBject) A graphics element, which can be any size that fits on the screen, that is moved as a unit by the blitter. BOBs are not limited in size like sprites, but because they are not independent of the background, are slower to move and require more programming effort to manipulate.

Copper: A part of the animation coprocessor chip that makes sure that the screen's image and the requests of the graphics chips stay in synchronization.

Font: All the characters in a typeface. The Amiga's system software has seven built-in fonts; its default font is named Topaz.

Genlock: (Amiga Genlock 1300) A device that allows the Amiga's graphics to be combined and synchronized with those from an external video source, such as a television or VCR.

Hold-and-Modify: (HAM) A graphics display mode in which the Amiga can display all possible 4,096 colors on its screen at once.

ILBM: (InterLeaved BitMap) A two-dimensional raster or bit-mapped image, with color. ILBM is one of the forms supported within the IFF standard; therefore pictures stored in ILBM format can be transferred from one application to another.

Interlacing: The method used by the Amiga to display 400 horizontal lines on the screen: i.e., high resolution mode. Interlacing causes the Amiga's hi-res flicker noticeable on many monitors. The standard Amiga display mode is non-interlaced, displaying 200 lines horizontally.

Pixels: (From picture elements) The dots of light that make up the computer display. Screen resolution is given in terms of pixels vertically and horizontally, e.g., 320 × 200 (low res).

Raster Image: A picture made up of individual dots or pixels. This is as opposed to a vector image, which is made up from lines. The Amiga produces raster images on its monitor.

Resolution: The number of pixels defined vertically and horizontally. The Amiga has five bit-mapped resolutions. These come from two horizontal resolutions (200 and 400) times two vertical resolutions (320 and 640) plus a special Hold-and-Modify mode.

RGB: (Red-Green-Blue) A term used to describe digital and analog color monitors. RGB digital monitors can display 16 colors; RGB analog monitors (Amiga 1080) can theoretically display an unlimited number.

Sprite: A type of graphics element that is treated as a unit by the Amiga's special graphics hardware, and that is independent of the background. (The Amiga's pointer is a Sprite.)

Video Coprocessor Chip: Nicknamed *Denise*, this coprocessor assists the 68000 in many of the screen control funtions; it handles the screen display and supports sprites.

Amiga Hardware Registers

Name		ADD	R/W	Chip	Function
BLTDDAT	&*	000	ER	A	Blitter destination early read (dummy address)
DMACONR	*	002	R	AP	DMA control (and blitter status) read
VPOSR	*	004	R	A	Read vert. most significant bit (and frame flop)
VHPOSR	*	006	R	A	Read vert. and horiz. position of beam
DSKDATR	&*	008	ER	P	Disk data early read (dummy address)
JOY0DAT	*	00A	R	D	Controller 0 data (vert., horiz.)
JOY1DAT	*	00C	R	D	Controller 1 data (vert., horiz)
CLXDAT	*	00E	R	D	Collision data register (read and clear)
ADKCONR	*	010	R	P	Audio, disk control register read
POT0DAT	*	012	R	P	Pot counter pair 0 data (vert., horiz)
POT1DAT	*	014	R	P	Pot counter pair 1 data (vert., horiz)
POTGOR	*	016	R	P	Pot port data read (formerly POTINP)
SERDATR	*	018	R	P	Serial data port and status read
DSKBYTR	*	01A	R	P	Disk data byte and status read
INTENAR	*	01C	R	P	Interrupt enable bits read
INTREQR	*	01E	R	P	Interrupt request bits read
DSKPTH	+*	020	W	A	Disk pointer (high 3 bits)
DSKPTL	+*	022	W	A	Disk pointer (low 15 bits)
DSKLEN	*	024	W	P	Disk length
DSKDAT	&*	026	W	P	Disk DMA data write
REFPTR	&*	028	W	A	Refresh pointer
VPOSW	*	02A	W	A	Write vert. most significant bit (and frame flop)
VHPOSW	*	02C	W	A	Write vert. and horiz. position of beam
COPCON	*	02E	W	A	Coprocessor control register (CDANG)
SERDAT	*	030	W	P	Serial port data and stop bits write
SERPER	*	032	W	P	Serial port period and control
POTGO	*	034	W	P	Pot port data write and start
JOYTEST	*	036	W	D	Write to all four controller counters at once
STREQU	&*	038	S	D	Strobe for horiz. sync with VB and EQU
STRVBL	&*	03A	S	D	Strobe for horiz. sync with VB (vert. blank)
STRHOR	&*	03C	S	DP	Strobe for horiz. sync
STRLONG	&*	03E	S	D	Strobe for identification of long horiz. line
BLTCON0	~	040	W	A	Blitter control register 0
BLTCON1	~	042	W	A	Blitter control register 1
BLTAFWM	~	044	W	A	Blitter first word mask for source A
BLTALWM	~	046	W	A	Blitter last word mask for source A
BLTCPTH	+~	048	W	A	Blitter pointer to source C (high 3 bits)
BLTCPTL	+~	04A	W	A	Blitter pointer to souce C (low 15 bits)
BLTBPTH	+~	04C	W	A	Blitter pointer to souce B (high 3 bits)
BLTBPTL	+~	04E	W	A	Blitter pointer to souce B (low 15 bits)
BLTAPTH	+~	050	W	A	Blitter pointer to source A (high 3 bits)
BLTAPTL	+~	052	W	A	Blitter pointer to source A (low 15 bits)
BLTDPTH	+~	054	W	A	Blitter pointer to destination D (high 3 bits)
BLTDPTL	+~	056	W	A	Blitter pointer to destination D (low 15 bits)
BLTSIZE	~	058	W	A	Blitter start and size (window width, height)
	~	05A			
	~	05C			
	~	05E			
BLTCMOD	~	060	W	A	Blitter modulo for source C
BLTBMOD	~	062	W	A	Blitter modulo for source B
BLTAMOD	~	064	W	A	Blitter modulo for source A
BLTDMOD	~	066	W	A	Blitter modulo for destination D
	~	068			
	~	06A			
	~	06C			
	~	06E			
BLTCDAT	%~	070	W	A	Blitter source C data register
BLTBDAT	%~	072	W	A	Blitter source B data register
BLTADAT	%~	074	W	A	Blitter source A data register
	~	076			
	~	078			
	~	07A			
	~	07C			
DSKSYNC	~	07E	R	P	Disk sync pattern register for disk read
COP1LCH	+	080	W	A	Coprocessor first location reg. (high 3 bits)

Name		ADD	R/W	Chip	Function
COP1LCL	+	082	W	A	Coprocessor first location reg. (low 15 bits)
COP2LCH	+	084	W	A	Coprocessor 2nd location reg. (high 3 bits)
COP2LCL	+	086	W	A	Coprocessor 2nd location reg. (low 14 bits)
COPJMP1		088	S	A	Coprocessor restart at first location
COPJMP2		08A	S	A	Coprocessor restart at second location
COPINS		08C	W	A	Coprocessor instruction fetch identity
DIWSTRT		08E	W	A	Display window start (upper left v/h pos.)
DIWSTOP		090	W	A	Display window stop (lower right v/h pos.)
DDFSTRT		092	W	A	Display bit plane data fetch start (horiz. pos.)
DDFSTOP		094	W	A	Display bit plane data fetch stop (horiz. pos.)
DMACON		096	W	ADP	DMA control write (clear or set)
CLXCON		098	W	D	Collision control
INTENA		09A	W	P	Interrupt enable bits (clear or set bits)
INTREQ		09C	W	P	Interrupt request bits (clear or set bits)
ADKCON		09E	W	P	Audio, disk, UART control
AUD0LCH	+	0A0	W	A	Audio channel 0 location (high 3 bits)
AUD0LCL	+	0A2	W	A	Audio channel 0 location (low 15 bits)
AUD0LEN		0A4	W	P	Audio channel 0 length
AUD0PER		0A6	W	P	Audio channel 0 period
AUD0VOL		0A8	W	P	Audio channel 0 volume
AUD0DAT	&	0AA	W	P	Audio channel 0 data
		0AC			
		0AE			
AUD1LCH	+	0B0	W	A	Audio channel 1 location (high 3 bits)
AUD1LCL	+	0B2	W	A	Audio channel 1 location (low 15 bits)
AUD1LEN		0B4	W	P	Audio channel 1 length
AUD1PER		0B6	W	P	Audio channel 1 period
AUD1VOL		0B8	W	P	Audio channel 1 volume
AUD1DAT	&	0BA	W	P	Audio channel 1 data
		0BC			
		0BE			
AUD2LCH	+	0C0	W	A	Audio channel 2 location (high 3 bits)
AUD2LCL	+	0C2	W	A	Audio channel 2 location (low 15 bits)
AUD2LEN		0C4	W	P	Audio channel 2 length
AUD2PER		0C6	W	P	Audio channel 2 period
AUD2VOL		0C8	W	P	Audio channel 2 volume
AUD2DAT	&	0CA	W	P	Audio channel 2 data
		0CC			
		0CE			
AUD3LCH	+	0D0	W	A	Audio channel 3 location (high 3 bits)
AUD3LCL	+	0D2	W	A	Audio channel 3 location (low 15 bits)
AUD3LEN		0D4	W	P	Audio channel 3 length
AUD3PER		0D6	W	P	Audio channel 3 period
AUD3VOL		0D8	W	P	Audio channel 3 volume
AUD3DAT	&	0DA	W	P	Audio channel 3 data
		0DC			
		0DE			
BPL1PTH	+	0E0	W	A	Bit plane 1 pointer (high 3 bits)
BPL1PTL	+	0E2	W	A	Bit plane 1 pointer (low 15 bits)
BPL2PTH	+	0E4	W	A	Bit plane 2 pointer (high 3 bits)
BPL2PTL	+	0E6	W	A	Bit plane 2 pointer (low 15 bits)
BPL3PTH	+	0E8	W	A	Bit plane 3 pointer (high 3 bits)
BPL3PTL	+	0EA	W	A	Bit plane 3 pointer (low 15 bits)
BPL4PTH	+	0EC	W	A	Bit plane 4 pointer (high 3 bits)
BPL4PTL	+	0EE	W	A	Bit plane 4 pointer (low 15 bits)
BPL5PTH	+	0F0	W	A	Bit plane 5 pointer (high 3 bits)
BPL5PTL	+	0F2	W	A	Bit plane 5 pointer (low 15 bits)
BPL6PTH	+	0F4	W	A	Bit plane 6 pointer (high 3 bits)
BPL6PTL	+	0F6	W	A	Bit plane 6 pointer (low 15 bits)
		0F8			
		0FA			
		0FC			
		0FE			
BLPCON0		100	W	AD	Bit plane control register (misc. control bits)
BPLCON1		102	W	D	Bit plane control reg. (scroll value PF1, PF2)
BPLCON2		104	W	D	Bit plane control reg. (priority control)
		106			
BPL1MOD		108	W	A	Bit plane modulo (odd planes)

Name		ADD	R/W	Chip
BPL2MOD		10A	W	A
		10C		
		10E		
BPL1DAT	&	110	W	D
BPL2DAT	&	112	W	D
BPL3DAT	&	114	W	D
BPL4DAT	&	116	W	D
BPL5DAT	&	118	W	D
BPL6DAT	&	11A	W	D
		11C		
		11E		
SPR0PTH	+	120	W	A
SPR0PTL	+	122	W	A
SPR1PTH	+	124	W	A
SPR1PTL	+	126	W	A
SPR2PTH	+	128	W	A
SPR2PTL	+	12A	W	A
SPR3PTH	+	12C	W	A
SPR3PTL	+	12E	W	A
SPR4PTH	+	130	W	A
SPR4PTL	+	132	W	A
SPR5PTH	+	134	W	A
SPR5PTL	+	136	W	A
SPR6PTH	+	138	W	A
SPR6PTL	+	13A	W	A
SPR7PTH	+	13C	W	A
SPR7PTL	+	13E	W	A
SPR0POS	%	140	W	AD
SPR0CTL	%	142	W	AD
SPR0DATA	%	144	W	D
SPR0DATB	%	146	W	D
SPR1POS	%	148	W	AD
SPR1CTL	%	14A	W	AD
SPR1DATA	%	14C	W	D
SPR1DATB	%	14E	W	D
SPR2POS	%	150	W	AD
SPR2CTL	%	152	W	AD
SPR2DATA	%	154	W	D
SPR2DATB	%	156	W	D
SPR3POS	%	158	W	AD
SPR3CTL	%	15A	W	AD
SPR3DATA	%	15C	W	D
SPR3DATB	%	15E	W	D
SPR4POS	%	160	W	AD
SPR4CTL	%	162	W	AD
SPR4DATA	%	164	W	D
SPR4DATB	%	166	W	D
SPR5POS	%	168	W	AD
SPR5CTL	%	16A	W	AD
SPR5DATA	%	16C	W	D
SPR5DATB	%	16E	W	D
SPR6POS	%	170	W	AD
SPR6CTL	%	172	W	AD
SPR6DATA	%	174	W	D
SPR6DATB	%	176	W	D
SPR7POS	%	78	W	AD
SPR7CTL	%	17A	W	AD
SPR7DATA	%	17C	W	D
SPR7DATB	%	17E	W	D
COLOR00		180	W	D
COLOR01		182	W	D
COLOR02		184	W	D
COLOR03		186	W	D
COLOR04		188	W	D
COLOR05		18A	W	D
COLOR06		18C	W	D
COLOR07		18E	W	D
COLOR08		190	W	D

Function
Bit plane modulo (even planes)
Bit plane 1 data (parallel-to-serial convert)
Bit plane 2 data (parallel-to-serial convert)
Bit plane 3 data (parallel-to-serial convert)
Bit plane 4 data (parallel-to-serial convert)
Bit plane 5 data (parallel-to-serial convert)
Bit plane 6 data (parallel-to-serial convert)
Sprite 0 pointer (high 3 bits)
Sprite 0 pointer (low 15 bits)
Sprite 1 pointer (high 3 bits)
Sprite 1 pointer (low 15 bits)
Sprite 2 pointer (high 3 bits)
Sprite 2 pointer (low 15 bits)
Sprite 3 pointer (high 3 bits)
Sprite 3 pointer (low 15 bits)
Sprite 4 pointer (high 3 bits)
Sprite 4 pointer (low 15 bits)
Sprite 5 pointer (high 3 bits)
Sprite 5 pointer (low 15 bits)
Sprite 6 pointer (high 3 bits)
Sprite 6 pointer (low 15 bits)
Sprite 7 pointer (high 3 bits)
Sprite 7 pointer (low 15 bits)
Sprite 0 vert.-horiz. start position data
Sprite 0 vert. stop position and control data
Sprite 0 image data register A
Sprite 0 image data register B
Sprite 1 vert.-horiz. start position data
Sprite 1 vert. stop position and control data
Sprite 1 image data register A
Sprite 1 image data register B
Sprite 2 vert.-horiz. start position data
Sprite 2 vert. stop position and control data
Sprite 2 image data register A
Sprite 2 image data register B
Sprite 3 vert.-horiz. start position data
Sprite 3 vert. stop position and control data
Sprite 3 image data register A
Sprite 3 image data register B
Sprite 4 vert.-horiz. start position data
Sprite 4 vert. stop position and control data
Sprite 4 image data register A
Sprite 4 image data register B
Sprite 5 vert.-horiz. start position data
Sprite 5 vert. stop position and control data
Sprite 5 image data register A
Sprite 5 image data register B
Sprite 6 vert.-horiz. start position data
Sprite 6 vert. stop position and control data
Sprite 6 image data register A
Sprite 6 image data register B
Sprite 7 vert.-horiz. start position data
Sprite 7 vert. stop position and control data
Sprite 7 image data register A
Sprite 7 image data register B
Color table 00
Color table 01
Color table 02
Color table 03
Color table 04
Color table 05
Color table 06
Color table 07
Color table 08

Name	ADD	R/W	Chip	Function
COLOR09	192	W	D	Color table 09
COLOR10	194	W	D	Color table 10
COLOR11	196	W	D	Color table 11
COLOR12	198	W	D	Color table 12
COLOR13	19A	W	D	Color table 13
COLOR14	19C	W	D	Color table 14
COLOR15	19E	W	D	Color table 15
COLOR16	1A0	W	D	Color table 16
COLOR17	1A2	W	D	Color table 17
COLOR18	1A4	W	D	Color table 18
COLOR19	1A6	W	D	Color table 19
COLOR20	1A8	W	D	Color table 20
COLOR21	1AA	W	D	Color table 21
COLOR22	1AC	W	D	Color table 22
COLOR23	1AE	W	D	Color table 23
COLOR24	1B0	W	D	Color table 24
COLOR25	1B2	W	D	Color table 25
COLOR26	1B4	W	D	Color table 26
COLOR27	1B6	W	D	Color table 27
COLOR28	1B8	W	D	Color table 28
COLOR29	1BA	W	D	Color table 29
COLOR30	1BC	W	D	Color table 30
COLOR31	1BE	W	D	Color table 31
RESERVED	1110X			
RESERVED	1111X			
NOOP(NULL)	1FE			

ADD—offset to base chip address $DFF000
&—register used by DMA channel only
%—register used by DMA channel usually, processors sometimes
+—address register pair. Low word uses DB1-DB15; high word uses DB0-DB2
*—address not writable by the copper
~—address not writable by the copper unless COPCON is set true
A,D,P—Agnus, Denise, Paula
R,W—Read, Write
ER—Early read
S—Strobe
PTL,PTH—18-bit DMA address pointer
LCL,LCH—18-bit starting address DMA pointer
MOD—15-bit modulo

Registers in the Peripheral Interface Adapters (8520s)

8520-A	8520-B	Name	Explanation
BFE001	BFD000	PRA	Peripheral data register A
BFE101	BFD100	PRB	Peripheral data register B
BFE201	BFD200	DDRA	Data direction register A
BFE301	BFD300	DDRB	Data direction register B
BFE401	BFD400	TALO	TIMER A low register
BFE501	BFD500	TAHI	TIMER A high register
BFE601	BFD600	TBLO	TIMER B low register
BFE701	BFD700	TBHI	TIMER B high register
BFE801	BFD800		Event LSB
BFE901	BFD900		Event 8-15
BFEA01	BFDA00		Event MSB
BFEB01	BFDB00		No connect
BFEC01	BFDC00	SDR	Serial data register
BFED01	BFDD00	ICR	Interrupt control register
BFEE01	BFDE00	CRA	Control register A
BFEF01	BFDF00	CRB	Control register B

Amiga 500/1000 86 Pin Expansion Bus

Pin	Function	Pin	Function	Pin	Function	Pin	Function	Pin	Function
1	GND	19	/EINT2	37	DGND	55	/HLT	73	DGND
2	GND	20	/PALOPE	38	A12	56	A20	74	/AS
3	GND	21	A5	39	A13	57	A22	75	PD0
4	GND	22	/EINT6	40	/IPL0	58	A21	76	PD10
5	+5V	23	A6	41	A14	59	A23	77	PD1
6	+5V	24	A4	42	/IPL1	60	/BR	78	PD9
7	N.C.	25	DGNB	43	A15	61	DGND	79	PD2
8	-5V	26	A3	44	/IPL2	62	/BGACK	80	PD8
9	N.C.	27	A2	55	A16	63	PD15	81	PD3
10	+12V	28	A7	46	/BERR	64	/BG	82	PD7
11	N.C.	29	A1	47	A17	65	PD14	83	PD4
12	/CONFIG	30	A8	48	/VPA	66	/DTACK	84	PD6
13	DGND	31	FC0	49	DGND	67	PD13	85	DGND
14	/C3	32	A9	50	E	68	/PRW	86	PD5
15	CDAC	33	FC1	51	/VMA	69	PD12		
16	/C1	34	A10	52	A18	70	/LDS		
17	/OVR	35	FC2	53	/RES	71	PD11		
18	XRDY	36	A11	54	A19	72	/UDS		

DIFFERENCES BETWEEN AMIGA 2000 CPU SLOT AND AMIGA 1000 EXPANSION BUS

9	28MHZ	20	N.C.

Parallel Ports

A-1000
D-25 male

Pin	Function	Pin	Function
1	DRDY*	14	GND
2	Data 0	15	GND
3	Data 1	16	GND
4	Data 2	17	GND
5	Data 3	18	GND
6	Data 4	19	GND
7	Data 5	20	GND
8	Data 6	21	GND
9	Data 7	22	GND
10	ACK*	23	+5v
11	BUSY (data)	24	NC
12	POUT (clk)	25	RESET*
13	SEL		

A500/A2000
D-25 female

Pin	Function	Pin	Function
1	STROBE*	14	+5v pullup
2	Data 0	15	NC
3	Data 1	16	RESET*
4	Data 2	17	GND
5	Data 3	18	GND
6	Data 4	19	GND
7	Data 5	20	GND
8	Data 6	21	GND
9	Data 7	22	GND
10	ACK*	23	GND
11	BUSY	24	GND
12	POUT	25	GND
13	SEL		

Amiga 2000 Zorro Expansion Specification

Pin	Function	Pin	Function	Pin	Function	Pin	Function	Pin	Function
1	GND	21	A5	41	A14	61	GND	81	BD3
2	GND	22	/EINT6	42	/EINT5	62	/BGACK	82	BD7
3	GND	23	A6	43	A15	63	BD15	83	BD4
4	GND	24	A4	44	/EINT4	64	/BGx	84	BD6
5	+5V	25	GND	45	A16	65	BD14	85	GND
6	+5V	26	A3	46	/BERR	66	/DTACK	86	BD5
7	*	27	A2	47	A17	67	BD13	87	GND
8	-5V	28	A7	48	/VPA	68	READ	88	GND
9	/SLAVEx	29	A1	49	GND	69	BD12	89	GND
10	+12V	30	A8	50	E	70	/BLDS	90	GND
11	**	31	BFC0	51	/VMA	71	BD11	91	GND
12	***	32	A9	52	A18	72	/BUDS	92	7ME
13	GND	33	BFC1	53	/RES	73	GND	93	DOE
14	/C3B	34	A10	54	A19	74	/BAS	94	/RESB
15	CDACB	35	BFC2	55	/HLT	75	BD0	95	# /BGIN
16	/C1B	36	A11	56	A20	76	BD10	96	/EINT1
17	/OVR	37	GND	57	A22	77	BD1	97	RESERV8
18	XRDY	38	A12	58	A21	78	BD9	98	RESERV9
19	/EINT2	39	A13	59	A23	79	BD2	99	GND
20	# -12V	40	/EINT7	60	/BRx	80	BD8	100	GND

* = /LOCAL-OWN ** = /CONFIG-OUTx *** = /CONFIG-INx
# = updated by Amiga

DIFFERENCES BETWEEN ORIGINAL ZORRO SPECIFICATION AND AMIGA 2000 SLOTS

19	/INT2	21	/INT6	42	/IPL1	92	7M
		40	/IPL0	44	/IPL2	96	RESERV7

Serial Ports

A1000
D-25 female

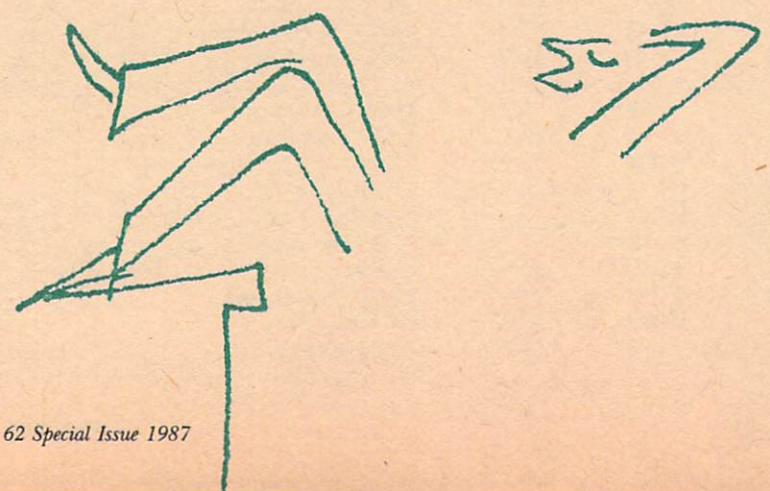
Pin	Function	Pin	Function
1	GND	14	-5Vdc
2	TxD	15	AUDO
3	RxD	16	AUDI
4	RTS	17	EB
5	CTS	18	INT2*
6	DSR	19	---
7	GND	20	DTR
8	DCD	21	+5Vdc
9	---	22	---
10	---	23	+12Vdc
11	---	24	C2*
12	---	25	RESB*
13	---		

A500/A2000
D-25 male

Pin	Function	Pin	Function
1	GND	14	---
2	TxD	15	---
3	RxD	16	---
4	RTS	17	---
5	CTS	18	AUDI
6	DSR	19	---
7	GND	20	DTR
8	DCD	21	---
9	+12v	22	RI
10	-12v	23	---
11	AUDO	24	---
12	---	25	---
13	---		

Memory Map

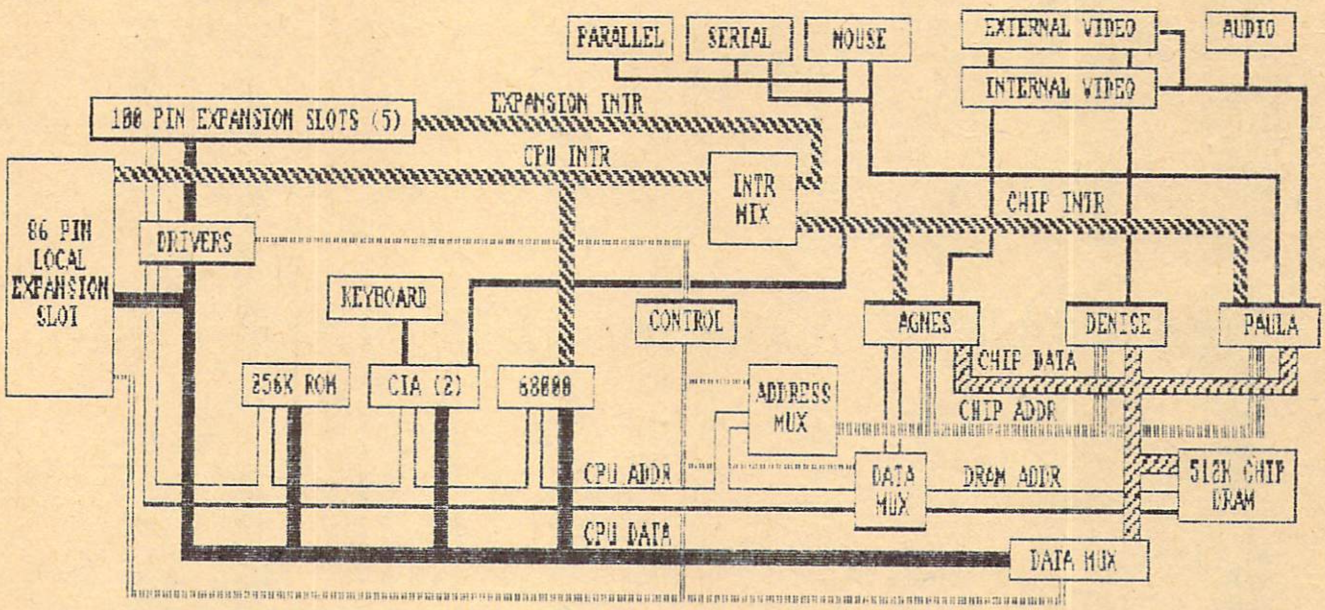
Address	Region
000000	512K Chip Memory
080000	Reserved
200000	8 Megabyte Memory Space For Auto-Configuration Expansion Devices
A00000	Reserved
BFD000	I/O For 8520's
C00000	CPU RAM
DFF000	Custom Chip Addresses
E00000	Reserved
E80000	Auto-Configuration Expansion Decoding
F00000	Reserved
F80000	ROM
FFFFFF	



A2000 System Block Diagram*

A1000 lacks 100-pin expansion circuitry

A500 has some expansion RAM on the same side of the bus as the chip RAM



* Courtesy Commodore Business Machines

ASCII Chart

Decimal	ASCII	HEX	Binary	Decimal	ASCII	HEX	Binary	Decimal	ASCII	HEX	Binary
0	NUL	00	00000000	29	GS	1D	00011101	58	:	3A	00111010
1	SOH	01	00000001	30	RS	1E	00011110	59	;	3B	00111011
2	STX	02	00000010	31	US	1F	00011111	60	<	3C	00111100
3	ETX	03	00000011	32	SP	20	00100000	61	=	3D	00111101
4	EOT	04	00000100	33	!	21	00100001	62	>	3E	00111110
5	ENQ	05	00000101	34	"	22	00100010	63	?	3F	00111111
6	ACK	06	00000110	35	#	23	00100011	64	@	40	01000000
7	BEL	07	00000111	36	$	24	00100100	65	A	41	01000001
8	BS	08	00001000	37	%	25	00100101	66	B	42	01000010
9	HT	09	00001001	38	&	26	00100110	67	C	43	01000011
10	LF	0A	00001010	39	'	27	00100111	68	D	44	01000100
11	VT	0B	00001011	40	(	28	00101000	69	E	45	01000101
12	FF	0C	00001100	41	)	29	00101001	70	F	46	01000110
13	CR	0D	00001101	42	*	2A	00101010	71	G	47	01000111
14	SO	0E	00001110	43	+	2B	00101011	72	H	48	01001000
15	SI	0F	00001111	44	,	2C	00101100	73	I	49	01001001
16	DLE	10	00010000	45	–	2D	00101101	74	J	4A	01001010
17	DC1	11	00010001	46	.	2E	00101110	75	K	4B	01001011
18	DC2	12	00010010	47	/	2F	00101111	76	L	4C	01001100
19	DC3	13	00010011	48	0	30	00110000	77	M	4D	01001101
20	DC4	14	00010100	49	1	31	00110001	78	N	4E	01001110
21	NAK	15	00010101	50	2	32	00110010	79	O	4F	01001111
22	SYN	16	00010110	51	3	33	00110011	80	P	50	01010000
23	ETB	17	00010111	52	4	34	00110100	81	Q	51	01010001
24	CAN	18	00011000	53	5	35	00110101	82	R	52	01010010
25	EM	19	00011001	54	6	36	00110110	83	S	53	01010011
26	SUB	1A	00011010	55	7	37	00110111	84	T	54	01010100
27	ESC	1B	00011011	56	8	38	00111000	85	U	55	01010101
28	FS	1C	00011100	57	9	39	00111001	86	V	56	01010110

Decimal	ASCII	HEX	Binary	Decimal	ASCII	HEX	Binary	Decimal	ASCII	HEX	Binary
87	W	57	01010111	102	f	66	01100110	117	u	75	01110101
88	X	58	01011000	103	g	67	01100111	118	v	76	01110110
89	Y	59	01011001	104	h	68	01101000	119	w	77	01110111
90	Z	5A	01011010	105	i	69	01101001	120	x	78	01111000
91	[	5B	01011011	106	j	6A	01101010	121	y	79	01111001
92	\	5C	01011100	107	k	6B	01101011	122	z	7A	01111010
93	]	5D	01011101	108	l	6C	01101100	123	{	7B	01111011
94	^	5E	01011110	109	m	6D	01101101	124	|	7C	01111100
95	_	5F	01011111	110	n	6E	01101110	125	}	7D	01111101
96	'	60	01100000	111	o	6F	01101111	126	~	7E	01111110
97	a	61	01100001	112	p	70	01110000	127	DEL	7F	01111111
98	b	62	01100010	113	q	71	01110001				
99	c	63	01100011	114	r	72	01110010				
100	d	64	01100100	115	s	73	01110011				
101	e	65	01100101	116	t	74	01110100				

PRINTER DEVICE COMMAND FUNCTIONS*

Name	Cmd No.	Escape Sequence	Function
aRIS	0	ESCc	reset
aRIN	1	ESC#1	initialize
aIND	2	ESCD	lf
aNEL	3	ESCE	return,lf
aRI	4	ESCM	reverse lf
aSGR0	5	ESC[0m	normal char set
aSGR3	6	ESC[3m	italics on
aSGR23	7	ESC[23m	italics off
aSGR4	8	ESC[4m	underline on
aSGR24	9	ESC[24m	underline off
aSGR1	10	ESC[1m	boldface on
aSGR22	11	ESC[22m	boldface off
aSFC	12	ESC[nm	set foreground color where n stands for a pair of ASCII digits, 3 followed by any number 0–9
aSBC	13	ESC[nm	set background color where n stands for a pair of ASCII digits, 4 followed by any number 0–9
aSHORP0	14	ESC[0w	normal pitch
aSHORP2	15	ESC[2w	elite on
aSHORP1	16	ESC[1w	elite off
aSHORP4	17	ESC[4w	condensed fine on
aSHORP3	18	ESC[3w	condensed off
aSHORP6	19	ESC[6w	enlarged on
aSHORP5	20	ESC[5w	enlarged off
aDEN6	21	ESC[6"z	shadow print on
aDEN5	22	ESC[5"z	shadow print off
aDEN4	23	ESC[4"z	doublestrike on
aDEN3	24	ESC[3"z	doublestrike off
aDEN2	25	ESC[2"z	NLQ on
aDEN1	26	ESC[1"z	NLQ off
aSUS2	27	ESC[2v	superscript on
aSUS1	28	ESC[1v	superscript off
aSUS4	29	ESC[4v	subscript on
aSUS3	30	ESC[3v	subscript off
aSUS0	31	ESC[0v	normalize the line
aPLU	32	ESCL	partial line up
aPLD	33	ESCK	partial line down
aFNT0	34	ESC(B	US char set
aFNT1	35	ESC(R	French char set

Name	Cmd No.	Escape Sequence	Function
aFNT2	36	ESC(K	German char set
aFNT3	37	ESC(A	UK char set
aFNT4	38	ESC(E	Danish I char set
aFNT5	39	ESC(H	Swedish char set
aFNT6	40	ESC(Y	Italian char set
aFNT7	41	ESC(Z	Spanish char set
aFNT8	42	ESC(J	Japanese char set
aFNT9	43	ESC(6	Norwegian char set
aFNT10	44	ESC(C	Danish II char set
aPROP2	45	ESC[2p	proportional on
aPROP1	46	ESC[1p	proportional off
aPROP0	47	ESC[0p	proportional clear
aTSS	48	ESC[n E	set proportional offset
aJFY5	49	ESC[5 F	auto left justify
aJFY7	50	ESC[7 F	auto right justify
aJFY6	51	ESC[6 F	auto full justify
aJFY0	52	ESC[0 F	auto justify off
aJFY3	53	ESC[3 F	letter space (justify)
aJFY1	54	ESC[1 F	word fill (auto center)
aVERP0	55	ESC[0z	1/8" line spacing
aVERP1	56	ESC[1z	1/6" line spacing
aSLPP	57	ESC[nt	set form length n
sPERF	58	ESC[nq	perf skip n (n≠0)
aPERF0	59	ESC[0q	perf skip off
aLMS	60	ESC#9	Left margin set
aRMS	61	ESC#0	Right margin set
aTMS	62	ESC#8	Top margin set
aBMS	63	ESC#2	Bottom margin set
aSTBM	64	ESC[n;nr	T&B margins
aSLRM	65	ESC[n;ns	L&R margin
aCAM	66	ESC#3	Clear margins
aHTS	67	ESCH	Set Horiz tab
aVTS	68	ESCJ	Set vertical tabs
aTBC0	69	ESC[0g	Clr horz tab
aTBC3	70	ESC[3g	Clear all h tab
aTBC1	71	ESC[1g	Clr vertical tabs
aTBC4	72	ESC[4g	Clr all v tabs
aTBCALL	73	ESC#4	Clr all h & v tabs
aTBSALL	74	ESC#5	Set default tabs
aEXTEND	75	ESC[n"x	Extended commands

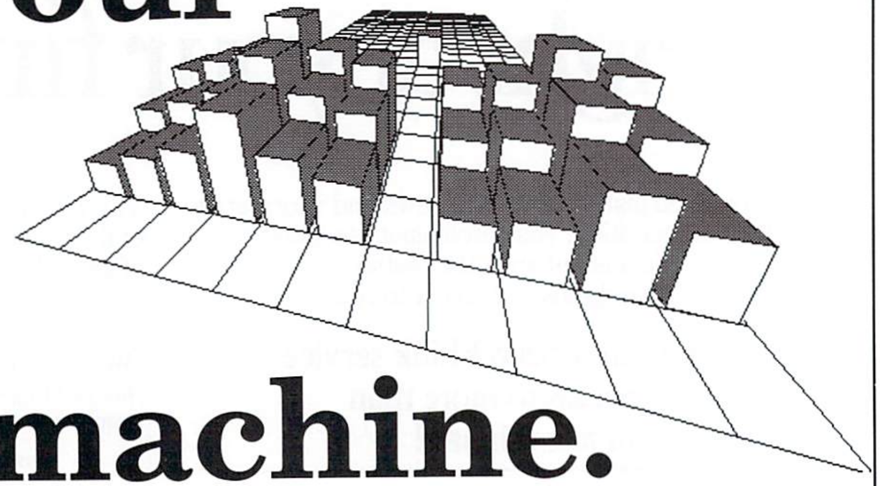
n stands for a decimal number expressed as a set of ASCII digits, for example 12.

The escape codes for use in Amiga Basic are in column 3.

* Table reprinted by permission of Addison-Wesley Publishing Company, Inc., Reading, MA 01867 from AMIGA ROM KERNEL REFERENCE MANUAL: LIBRARIES AND DEVICES © copyright 1986 by Commodore Electronics, Ltd. Publication date August 1986.

T R U E   B A S I C   L A N G U A G E   S Y S T E M

Turn your Amiga into a flying machine.



Speed, great graphics and enough built-in power to let you really fly: that's what you expect from your Amiga®. And that's what you get from programming in True BASIC.

It's a structured language that's easy to use. A compiler, editor and debugging tools rolled into one. It's the latest from Kemeny and Kurtz, the inventors of BASIC.

If you crunch numbers, you'll appreciate floating-point math that's very fast. Support for large, dynamic arrays, and built-in syntax for matrix algebra.

And True BASIC makes graphics even easier. Define your own coordinate schemes and color palettes. Plot entire arrays with a single statement. Apply built-in 2D transforms. Or use the 3D Graphics Library to put some depth in the picture.

With the Developer's Toolkit you can launch sprites and bobs. Add pull-down menus, and get at all the Intuition and Kernal features. But if portable code is important, there's True BASIC for the PC, Macintosh™ and Atari® ST as well.

As your programs get bigger, you'll want to build your own external libraries.

They can be separately compiled, in BASIC, C or assembly. Debug with breakpoints and immediate mode. Create keyboard macros to complement True BASIC's fantastic editor.

But if you're just learning how to fly, True BASIC will coach you with on- line HELP. Friendly error messages, and a tutorial-based User's Guide.

Find out why thousands of engineers, developers, and students use True BASIC for their flights of fancy. Visit your Amiga dealer, or call us today at 1-800-TRBASIC.

LANGUAGE
- multi-line functions
- SELECT CASE, IF/ELSE IF
- local, global variables
- recursion
- GKS graphics

LIMITS
- strings to 1 megabyte each
- floating-point 5e-309 to 1.8e+308
- unlimited program size, arrays, data
- names up to 31 characters
- auto memory management

COMPILER/DEBUGGER
- compiles to fast, compact b-code
- separately-compiled libraries
- parameter type-checking
- set breakpoints and continue
- immediate mode execution

EDITOR
- full-screen, mouse-based editor
- global search, replace
- block copy, move, delete
- menu, function key, command line interfaces all supported

Language System *$99.95*
Runtime Package *$99.95*

Developer's Toolkit, 3-D Graphics Library, Sorting & Searching, Advanced String Library
$49.95 each

Amiga is a registered trademark of Commodore-Amiga — Macintosh is licensed to Apple Computer — Atari is a registered trademark of Atari Corp — True BASIC is a trademark of True BASIC, Inc.

True BASIC inc.
39 South Main Street
Hanover, NH 03755
(603) 643-3882

Arrivals and Departures:

Input, Output and C

Learn to access printers, disk drives and other peripherals from your C programs.

By Vincent M. Hopson

In C programming, input from and output to anything other than the keyboard and screen (using *printf* or *scanf*) can be very confusing. Moreover, any errors in the specification of parameters to input/output (I/O) subroutines can cause the Amiga to warm start or hang without apparent cause. This article covers the basic aspects of I/O from C, and presents three similar programs to illustrate different methods of disk I/O.

When accessing a file in Amiga Basic, you use an OPEN statement to request a data stream, and supply a number to identify it. C uses the same concept, except the system supplies the file identifier and the open statement has three variations on the Amiga: *fopen*, when you want to use high-level file pointers (fp); *open* for using low-level file descriptors (fd); and Open, for using AmigaDOS file handles (fh). Both C and BASIC require that you further restrict access to the data stream by declaring a direction—read or write. AmigaDOS file-handle data streams are opened without a direction and are assumed to be both read and write.

I/O with AmigaDOS file handles (Open) is similar to using I/O with generic C file descriptors (*open*). The AmigaDOS function names begin with a capital letter to differentiate them from equivalent C subroutines.

Amiga Basic manipulates data streams by using PRINT# or INPUT# commands. The programmer supplies the number that identifies the desired stream. C allows the programmer more flexibility in the control of data flow. Some of the available data stream commands are *fput* (put a character), *fputs* (put a string) and *fprintf* (print a formatted string). Many C programs use *printf* and *scanf* for console I/O and the file pointer variations, *fprintf* and *fscanf*, for disk I/O. Another version of these same commands uses a string (an array of characters) as the destination of their activity: *sprintf* and *sscanf*. The string is then applied to the I/O data stream using *puts* or similar functions.

Closing the data stream before the program exits is generally required in BASIC, but C file pointers have no such restriction. When a C program exits, all of the file pointer data streams are closed automatically. However, C will not deallocate other resources such as file descriptors or file handles; therefore, it is good practice to make your own arrangements for closing files and returning resources.

A facet of I/O on the Amiga not to be covered in depth here is the system interface. Be aware, however, that many of these calls return useful information about files, such as creation or modification dates and file size.

Pointers, Descriptors and Handles

File pointers, descriptors and handles are the values that are returned when you open a file, depending upon the I/O method you choose. The values are required by the data stream handling functions in C.

Generally, file pointers are used with text-only files. If an executable file (one that contains values from 0x00 to 0xFF) is manipulated via file pointers rather than using *fread* or *fwrite*, NULLs present a problem. Since all strings are delimited with NULLs, this character cannot ►

appear inside a file pointer source or return buffer (string) as a valid character. To compound this problem, some of the file pointer I/O subroutines convert line-feeds into NULLs. Within these restrictions, file pointers are very powerful, since formatted printing or input can be done directly from the data stream instead of from a string buffer.

What you lose in flexibility with file pointers is regained with file descriptors and handles. All routines using these methods of I/O must buffer the input or output in a string and use string-manipulative routines to get the data to or from the data stream. Descriptors and handles are not as sensitive to the contents of the data stream as pointers, and may be used on any file. All of the routines that use them require the descriptor or handle, a buffer and the number of characters to get or put. If the routine returns a −1, an error has occurred. Any other number is the number of characters that were actually transferred. A zero indicates EOF (end of file).

File pointers, descriptors and handles are the values that are returned when you open a file, depending upon the I/O method you choose.

File handles point to AmigaDOS I/O structures, instead of acting as indices of I/O control blocks as file descriptors do. Handles and descriptors are used alike, except handle routines have the first letter capitalized and the file handle Open returns a NULL to represent failure instead of −1. All of the file descriptor and file handle routines with similar names have similar returns and functions. Take care to use file handles with routines that have capitalized names and file descriptors with uncapitalized routine names. Placing the wrong parameter in an I/O call can be catastrophic.

New C users should note that all of the routines that work with file pointers use file descriptors or handles to actually accomplish their task. This point is illustrated by the existence of a level two I/O call that returns the file descriptor associated with the file pointer data stream. This allows you to open a file with a pointer, acquire the descriptor associated with the same file and manipulate the stream with descriptor calls.

Descriptors and handles represent the lowest level—level one—of generic C I/O. Pointers represent level two I/O; subroutines that use pointers actually call routines on the next lower level (descriptors and handles) to perform their function. Called the layered approach, this technique is used frequently in large systems, such as the UNIX operating system. Each additional layer, or level, in a system requires you to know less and less about the specific implementation of the command you have executed. A layer, in general, should be less complicated than the layer beneath it and more complex than the one above. Generally, using high-level functions is easier than using low-level ones; it is also less efficient.

Keyboard interaction is a good example of the layered approach. A programmer executes a *scanf* command to obtain information for his program. A level two command, *scanf* uses the information in the subroutine call to request, via a level one command, a buffer full of data from the keyboard. The level one command calls a subroutine that actually manipulates the keyboard hardware interface to acquire characters until the user presses Return and the buffer is ready. The level one routine then terminates the buffer with a NULL and returns to level two. The *scanf* routine then processes the data into the chunks requested by the programmer in the control string and returns to the calling program. The programmer never needs to know how the keyboard processing system works, but is able to acquire data from it, which is why the layered approach is so useful.

AmigaDOS Devices

Nine devices exist under AmigaDOS: DF0:, RAM:, PRT:, PAR:, SER:, CON:, RAW:, * and NIL:. DF0: is the inboard disk drive driver; external drives take the names DF1:, DF2:, etc. RAM: is the RAMdisk device. PRT: is the generic printer driver, PAR: and SER: are the parallel and serial port drivers and CON: is the console driver. RAW: is a device that may be used to access keyboard input or console output without any translation of keystrokes by the CON: driver. The current window is *. NIL: is the proverbial bit bucket—anything sent to it will be dumped mercilessly into space.

You can access all of these devices with any version of the open command you wish to use. If your system-configuration file in the devs: directory is properly updated with Preferences, opening PRT: will access your printer whether it is on the serial or parallel port. The PRT: driver converts general Amiga function codes to the ones specifically for your printer. Available function codes for PRT: are outlined in the Technical Reference Guide included in this issue.

Opening some of these devices may require additional information. File names may be specified as DF0:*directory/file* or RAM:*directory/file*. CON: and RAW: opens must be declared as:

CON:*xpos/ypos/width/height/title*
RAW:*xpos/ypos/width/height/title*

Xpos and ypos specify where the window will appear. Width and height specify its size and title is the name at the titlebar.

Easy as 1, 2, 3. . .

Listing 1 is a program that uses file pointers to read a file from the disk and print it to the generic printer device, PRT:. Notice that opening a device is the same as opening a file; C considers all data streams as files. The first line of the generated printout contains the file name, followed by data from the file. This program is relatively simple and easy to expand upon.

Several checks are made to ensure the validity of the print request. A check of the argument count will determine if the user has input a file name. If no file name is present, a simple warning message is sent to the console and the program exits. A non-zero exit value alerts the calling program that an error has occurred and that normal processing should be suspended. The value of the return may also be used to classify the failure, and suggest cleanup or recovery work.

The program then tries to open a data stream to the printer, then to the requested file. If either attempt returns a NULL, the open was unsuccessful, and the program exits with an error code. Failure to open the printer reports the error only, whereas a failure to open the file for input must report the error *and* close the successfully opened printer data stream.

The *fprintf* statement sends a form feed to the printer, followed by Filename: and the name that was specified in the command line. Notice that the first parameter is the file pointer, followed by the control string and any parameters to convert.

Three file pointers are automatically opened when you execute a C program, *stdin* (standard input), *stdout* (standard output) and *stderr* (standard error output). The keyboard is usually *stdin*; *stdout* and *stderr* are typically the console. The *printf* command you use to print to the screen is just a specialized case of *fprintf*; it uses the console by default. Therefore the statement *printf(Test Statement);* is functionally equivalent to *fprintf(stdout, Test Statement);*. This also holds true for *scanf* and *fscanf*, except you'd of course substitute *stdin* for *stdout*. For error messages, use *stderr* to ensure that redirection from the command line will not cause the error message to disappear into some other file stream. I left this out of the examples to preserve simplicity.

The *while* loop gets lines from the input file with the *fgets* command. A *sizeof* argument provides the length of the buffer. You are then able to change the size of the buffer in the declaration at the top of the program without having to change the length parameter there. When the *fgets* subroutine returns a NULL, all of the characters from the file have been processed and an EOF condition exists. If you want, *rewind* will return to the top of the input file, and *lseek* returns to any byte in the file.

As long as *fgets* does not return a NULL, *fputs* sends the obtained data to the printer. This process continues until an EOF condition occurs to terminate the *while* statement. Both pointers are then closed, and the program ends.

There are more error conditions here than I have checked for. If the printer driver encounters some error, this routine will hang and not return. Unless you play with the on-line button randomly, or disconnect the printer, this condition is unlikely. You can either flag the situation as a problem in the documentation, or place an error check in the *while* loop. Whenever using I/O streams, it is good practice to plan for the unexpected and check all returns for validity. There are exceptions, however. If an error causes the Amiga to restart and you cannot prevent it, don't bother to check for that error. Since no recovery is possible, the error-checking code would probably never be executed.

Descriptors and File Handles

Listing 2 is the same program, using descriptors. The major differences are in the descriptor declarations and the inclusion of a character count variable. File descriptors are declared as integers, since they are indices into an array of I/O blocks.

A file called fcntl.h is also included to define the symbols O_RDONLY and O_WRONLY. These symbols are used to declare the direction of the I/O data stream; just as r and w were in the *fopen* statements of the previous example. AmigaDOS file handle Opens use MODE OLDFILE and MODE_NEWFILE definitions without regard to direction. These definitions may be found in libraries/dos.h.

The number of characters input by the read statement is stored in count. A tally is maintained to tell the write statement how much of the buffer is real data. Without this information, it is possible to print parts of the input buffer that are invalid. Both the printer file stream and file data stream are closed as the program exits.

Listing 3 is an example using file handles. File handles follow the conventions of pointers during opening, and descriptors in their use.

These three programs represent variations on a simple printing routine. The routine is prone to error, but does illustrate the use of C I/O. File pointers, handles and descriptors may be used together as well. For example, a file pointer could open the text data file to be printed and a file descriptor or handle could send data to the printer. This configuration is useful if you decide to take complete control of the output device and handle the special printer function codes yourself.

Whenever using I/O streams, it is good practice to plan for the unexpected and check all returns for validity. There are exceptions, however.

Programs and Signals

Finally, the signals CTRL-C, CTRL-D, CTRL-E and CTRL-F affect certain flags while a program is executing. The significance of this is not apparent until a command goes berserk and you wish to stop it without restarting the machine. If an application program periodically checks its signals, it is possible to cause the program to take some alternate action (such as terminating normally) by using signal flags. After a little experimentation, I found that the CTRL-C through CTRL-F flags were bits 12, 13, 14 and 15 of the signal longword stored in the task control block. All programs receive these signals whenever the CTRL-letter combination is used.

Listing 4 illustrates this in a template form. A FindTask call without a name parameter will return a pointer to this task's control block. The appropriate signal flags are checked by performing a logical AND comparison of the longword with a mask of signals that you wish to check for. If the signal is present, the result will be non-zero. AmigaDOS never clears these signals; you must do this yourself if a signal may occur more than once (i.e., if a signal doesn't terminate the program). Signal masks are defined in the include file libraries/dos.h and appear as SIGBREAKF_CTRLx. See the Libraries and Devices reference manual page D–170 lines 154 to 157.

This article is intended to get you started using C input/ouput. *The AmigaDOS Manual, Second Edition* by Commodore-Amiga is a helpful reference for some of the finer points of I/O in C on the Amiga. However, the best way to learn about I/O is to actually write some code and test it. The worst hazard of a program gone ►

bananas is an Amiga warm-start or the printer spewing bizarre patterns on a few sheets of paper. With so little to lose and so much to gain, experiment! ■

Vincent Hopson works as a software engineer for Spectrum Digital in Herndon, VA. Write to him c/o AmigaWorld *editorial.*

Listing 1. *File pointer example.*

```
/**************************************************************
 *                                                            *
 *      F I L E   P O I N T E R   E X A M P L E               *
 *                                                            *
 * This program will take the first argument from             *
 * the command line as a filename, and print it.              *
 *                                                            *
 **************************************************************/
#include "lattice/stdio.h"
/*
 * Error return codes.
 */
#define INSUFFICIENT_ARGS      1
#define NO_PRINTER             2
#define BAD_FILENAME           3

/**************************************************************
 *                                                            *
 *        M a i n   E n t r y   P o i n t                     *
 *                                                            *
 **************************************************************/
main(argc, argv)
int  argc;
char *argv[];
{
   char buffer[160]; /* temporary storage buffer  */
   FILE *fp;       /* pointer into file stream  */
   FILE *pfp;            /* pointer to printer stream */

/*
 * See if enough parameters are on the command line by
 * checking the number of parameters entered.
 */
   if (argc < 2)
   {
    /*
     * Not enough arguments.  Remind him of the usage,
     * and exit with appropriate return.
     */
      printf("Usage: print <filename>\n");
      exit(INSUFFICIENT_ARGS);
   }

   /*
    * Enough arguments are there.  Attempt to open the
    * printer data stream as a write stream.
    */
   pfp = fopen("PRT:", "w");
   if (pfp == NULL)
   {
      /*
       * Couldn't do it for some reason.  Tell the user,
       * and return error code.
       */
      printf("Cannot open printer data stream.\n");
      exit(NO_PRINTER);
   }
 /*
    * The printer is ready.  Attempt to open the specified
    * file on the command line.
    */
   fp = fopen(argv[1], "r");
   if (fp == NULL)
   {
    /*
     * Cannot open the file.  Close the printer data
     * stream, alert the user, and return code.
     */
      printf("Cannot open file data stream: %s\n",argv[1]);
      fclose(pfp);
      exit(BAD_FILENAME);
   }

   /*
    * Send a formfeed, and filename banner.
    */
   fprintf(pfp, "\fFilename: %s\n\n", argv[1]);

   /*
    * Loop here in the "while" construct.  Fill the
    * buffer with fgets, and dump it to the printer
    * with fputs.  Continue to loop here until NULL
    * characters are gotten (EOF).
    */
   while (fgets(buffer, sizeof(buffer), fp) != NULL)
      fputs(buffer, pfp);

   /*
    * Return all resources before exit.
    */
   fclose(pfp);
   fclose(fp);
}
```

Listing 2. *File descriptor example.*

```
/**************************************************************
 *                                                            *
 *   F I L E   D E S C R I P T O R   E X A M P L E            *
 *                                                            *
 *   This file performs the same task as the previous         *
 * example, and is commented where they differ.               *
 *                                                            *
 **************************************************************/
/*
 * "fcntl.h" contains definitions for O_RDONLY, and
 * O_WRONLY.
 */
#include "lattice/stdio.h"
#include "lattice/fcntl.h"
#define INSUFFICIENT_ARGS      1
#define NO_PRINTER             2
#define BAD_FILENAME           3
main(argc, argv)
int  argc;
char *argv[];
{
   char buffer[160];
   int fd;              /* input file descriptor       */
   int pfd;             /* printer file descriptor     */
   int count;           /* character count for buffer */

   if (argc < 2)
   {
      printf("Usage: print <filename>\n");
      exit(INSUFFICIENT_ARGS);
   }
   /*
    * Notice that the open contains O_WRONLY instead of
    * "w", and that a bad return is not zero, but -1.
    */
   pfd = open("PRT:", O_WRONLY);
   if (pfd == -1)
   {
      printf("Cannot open printer data stream.\n");
```

Listing continued on p. 72

Circle 309 on Reader Service card.

from p. 70

```
      exit(NO_PRINTER);
   }
   /*
    * Here, too, notice that the open contains O_RDONLY
    * instead of "r".
    */
   fd = open(argv[1], O_RDONLY);
   if (fd == -1)
   {
      printf("Cannot open file data stream: %s\n", argv[1]);
      close(pfd);
      exit(BAD_FILENAME);
   }

   /*
    * Major differences occur when using file descriptors
    * and pointers.  The string that we wish to place at
    * the top of the generated listing must be "sprintf"ed
    * into a buffer, then written to the data stream with
    * a "write" command.  The stclen command inside of the
    * write invocation will compute the size of the string
    * to be printed, since the filename may vary in size.
    */
   sprintf(buffer, "\fFilename: %s\n\n", argv[1]);
   write(pfd, buffer, stclen(buffer));

   /*
    * "count" stores the number of characters that were
    * moved in the read command.  This tells us the number
    * of valid characters in the buffer.  A sizeof statement
    * is used to send the size of the buffer.  If, in the
    * future we wish to change the size of "buffer", the
    * change will be reflected here without having to come
    * to this statement to change anything.
    *
    * I assume that the "write" command works properly.  An
    * error check should be placed there.
    */
   while ((count = read(fd, buffer, sizeof(buffer))) > 0)
      write(pfd, buffer, count);

   close(pfd);
   close(fd);
}
```

***Listing 3.** File handle example.*

```
/**********************************************************
 *                                                        *
 *      F I L E   H A N D L E   E X A M P L E             *
 *                                                        *
 **********************************************************/
/*
 * "dos.h" is included to define "MODE_xxxFILE", and
 * define the FileHandle structure.
 */
#include "libraries/dos.h"
#include "lattice/stdio.h"

#define INSUFFICIENT_ARGS      1
#define NO_PRINTER             2
#define BAD_FILENAME           3

/*
 * This definition will cause the compiler to handle
 * returns from the AmigaDOS "Open" statement properly,
 * and return the proper object.
 */
extern struct FileHandle *Open();

main(argc, argv)
int  argc;
char *argv[];
{
   char buffer[160];
   struct FileHandle *fh;      /* input file handle      */
   struct FileHandle *pfh;     /* printer file handle    */
   int count;

   if (argc < 2)
   {
      printf("Usage: print <filename>\n");
      exit(INSUFFICIENT_ARGS);
   }
   /*
    * Here the open statement is capitalized, and the
    * stream is declared as an existing file.  If a new
    * file was to be created, MODE_OLDFILE would be changed
    * to MODE_NEWFILE.
    * Notice also that an error condition is a NULL, not -1.
    */
   pfh = Open("PRT:", MODE_OLDFILE);
   if (pfh == NULL)
   {
      printf("Cannot open printer data stream.\n");
      exit(NO_PRINTER);
   }

   fh = Open(argv[1], MODE_OLDFILE);
   if (fh == NULL)
   {
      printf("Cannot open file data stream: %s\n", argv[1]);
      close(pfh);
      exit(BAD_FILENAME);
   }

   /*
    * Basically, the only difference here is that the
    * operation commands "Write", and "Read" have the first
    * letter capitalized.
    */
   sprintf(buffer, "\fFilename: %s\n\n", argv[1]);
   Write(pfh, buffer, stclen(buffer));

   while ((count = Read(fh, buffer, sizeof(buffer))) > 0)
      Write(pfh, buffer, count);

   Close(pfh);
   Close(fh);
}
```

***Listing 4.** Signal use program template.*

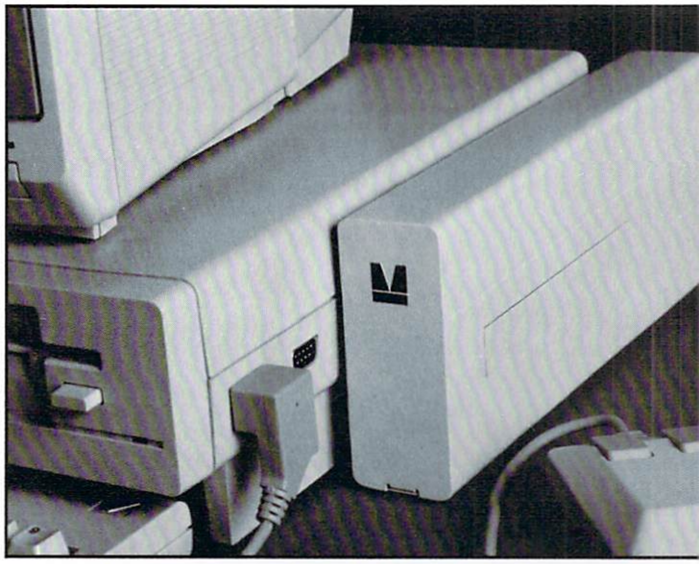
```
SIGNAL USE PROGRAM TEMPLATE

#include "exec/types.h"
#include "exec/tasks.h"
#include "libraries/dos.h"

extern struct Task *FindTask( );

main( )
{
     struct Task *taskcb;

     ... Other declarations, and program initialization

     taskcb = FindTask(" ");

     ... and to periodically check the flags ...

     if (tcb->tc_SigRecvd & (flags you wish to check))

     ... code to execute when signal arrives ...

     }
}
```

AVAILABLE NOW!

StarBoard2

If you've owned your Amiga® for a while now, you *know* you definitely need more than 512k of memory. You probably need *at least* double that amount...but you might need as much as an additional two megabytes. We want to urge you to use **StarBoard2** as the solution to your memory expansion problem –and to some of your other Amiga-expansion needs as well!

Auto-Configuring Fast RAM
Zero Wait States
User Expandable from 512k to 2 Megabytes
Bus Pass-Through
MultiFunction Option: battery/ clock, FPU, parity, Sticky-Disk



It's small, but it's BIG–

Since most of you want to expand your Amiga's memory without having to also expand your computer table, we designed **StarBoard2** and its two optional "daughterboards" to fit into a sleek, unobtrusive Amiga-styled case that snugly fastens to your computer with two precision-machined jackscrews.

The sculpted steel case of **StarBoard2** measures only 1.6" wide by 4.3" high by 10.2" long. You can access the inside of the case by removing just two small screws on the bottom and pulling it apart. We make **StarBoard2** easy to get into so that you or your dealer can expand it by installing up to one megabyte of RAM on the standard **StarBoard2** or up to two megabytes by adding in an Upper Deck.

This card has decks!

The basic **StarBoard2** starts out as a one megabyte memory space with 0k, 512k, or one megabyte installed. If you add in an optional **Upper Deck** (which plugs onto the Main Board inside the case) you bring **StarBoard2** up to its full two megabyte potential. You can buy your **StarBoard2** with the Upper Deck (populated or unpopulated) or buy the Upper Deck later as your need for memory grows.

And you can add other functions to **StarBoard2** by plugging in its second optional deck –the Multifunction Module!

StarBoard2: functions five!

If we count Fast Memory as one function, the addition of the **MultiFunction Module** brings the total up to five!

THE CLOCK FUNCTION:

Whenever you boot your Amiga you have to tell it what time it is! Add a MultiFunction Module to your **StarBoard2** and you can hand that tedious task to the battery-backed, real-time clock/calendar. A small piece of MicroBotics software in your WorkBench Startup-Sequence reads the clock and automatically sets the time and date in your Amiga. And the battery *is* included (we designed it to use an inexpensive, standard AAA battery which will last at least two years before needing replacement).

THE FLOATING POINT FUNCTION:

If any one aspect most characterizes the Amiga it's *fast* graphics! Most graphic routines make heavy use of the Amiga Floating Point Library. Replacing this library with the one we give you with your MultiFunction Module and installing a separately purchased Motorola 68881 FPU chip in the socket provided by the Module will speed up these math operations from 5 to 40 times! And if you write your own software, you can directly address this chip for increased speed in integer arithmetic operations in addition to floating point math.

THE PARITY CHECKING FUNCTION:

If you install an additional ninth RAM chip for every eight in your **StarBoard2**, then you can enable *parity checking*. Parity checking will alert you (with a bus-error message) in the event of any data corruption in **StarBoard2**'s memory space. So what good is it to know that your data's messed up if the hardware can't fix it for you? It will warn you against saving that data to disk and possibly destroying your database or your massive spreadsheet. The more memory you have in your system the more likely it is, statistically, that random errors will occur. Parity checking gives you some protection from this threat to your data residing in Fast RAM. Note that the Amiga's "chip" RAM cannot be parity checked.

THE IMMORTAL MEMORY DISK FUNCTION (STICKY-DISK):

When you've got a lot of RAM, you can make nice big RAM-Disks and speed up your Amiga's operations a lot! But there's one bad thing about RAM-Disks: they go away when you re-boot your machine. Sticky-Disk solves that problem for you. It turns all of the memory space inside a single **StarBoard2** into a Memory Disk that will survive a warm-reboot! When your Amiga attempts to grab a **StarBoard2** in Sticky-Disk mode, a hardware signal prevents the system from acquiring the **StarBoard2** as FastRAM (and thereby erasing your files) –instead it is re-recognized as a Memory Disk and its contents are preserved intact. If you want to work rapidly with large files of data that are being constantly updated (such as when developing software) you can appreciate the Sticky-Disk!

Fast RAM –no waiting!

StarBoard2 is a *totally* engineered product. It is a ZERO WAIT-STATE design, auto-configuring under AmigaDOS 1.2 as Fast RAM. Since AmigaDOS 1.1 doesn't support autoconfiguration, we also give you the software to configure memory in 1.1.

Any applications software which "looks" for Fast RAM will "find" **StarBoard2.** And you'll find that your applications run more efficiently due to **StarBoard2** on the bus.

A passing bus? Indeed!

What good is an Expansion Bus if it hits a dead end, as with some memory cards? Not much, we think –that's why we carefully and compatibly passed through the bus so you could attach other devices onto your Amiga (including another **StarBoard2**, of course!).

The sum of the parts...

A really nice feature of the **StarBoard2** system is that you can buy exactly what you need now without closing off your options for future exapansion. You can even buy a 0k **StarBoard2** (with a one megabyte capacity) and populate it with your own RAM (commonly available 256k by 1 by 150ns memory chips). When you add **StarBoard2** to your Amiga you have a powerful hardware combination, superior to any single-user micro on the market. See your Authorized Amiga Dealer today and ask for **StarBoard2**

SUGGESTED RETAIL PRICING:

StarBoard2, 0k (1 meg space):	$349
StarBoard2, 0k (2 meg space):	$395
StarBoard2, 512k (1 meg space):	$495
StarBoard2, 1 meg (1 meg space)	$595
StarBoard2, 2 megs installed:	$879
StarBoard2, 2 megs & MultiFunction:	$959
Upper Deck, 0k (1 meg space):	$ 99
MultiFunction Module:	$ 99
also available:	
Standard 256k memory card:	$129
MAS-Drive20, 20 meg harddisk:	$1495
MouseTime, mouseport clock:	$ 50

AMIGA is a registered trademark of Commodore-Amiga

MicroBotics, Inc.
811 Alpha Drive, Suite 335, Richardson, Texas 75081 / (214) 437-5330

Circle 325 on Reader Service card.

Where Things Get Done: An Overview of the Workbench

The Amiga's graphic interface lets you perform miracles with your mouse. No experience necessary.

By Vinoy Laughner

Workbench is the Amiga's distinctive graphics system of menus, windows and icons. It was designed appropriately for the Amiga's multitasking environment since it lets you visually keep charge of and control a number of things that are happening "simultaneously" and places a wide range of decisions and activities within your easy grasp. Sometimes the CLI is more direct, but Workbench, because of its non-linear, visual approach, can make complex operations as simple as 1 . . . 2 . . . clicks of the mouse, that is. Being no lover of programming and no willing initiate into DOS mysteries, when I go to work on the Amiga, I usually go first to Workbench.

This article is a *user's* introduction and overview of Workbench. It is intended to be a compact, overall reference; for more detail, refer to your *Introduction to Amiga* manual and the manual addition that comes with the version 1.2 Enhancer. This article presupposes that you have the version 1.2 upgrade; if you don't, it should be your next purchase. And make at least one backup copy of your Workbench disk for safe keeping (Kickstart and Extras also), because "things" can happen, and do.

Workbench Terminology

Certain objects and activities have special names in Workbench and are used throughout this article. They are quite easy to grasp and remember. Once they are understood, following through a process will be much simpler. Here's a short list:

Click: This means to press down a mouse button.

Double-click: This refers to clicking twice with the *left* mouse button. This is the usual method of opening Workbench windows and of activating programs by their icons.

Select: This means to click *once* on something with the *left* mouse button. Once selected, objects are ready to have some operation performed on them.

Highlighted: When objects (such as icons) have been selected, they change color, or are highlighted.

Drag: This means to select something (such as an icon or window) and while *continuing to hold the left mouse button down*, moving it to a different location.

Gadget: This refers to any box or thing upon which you are required to click the mouse to perform some operation, make a selection or yes/no-type decision.

Requester: These are gadgets in which you must type something in order to proceed with some operation.

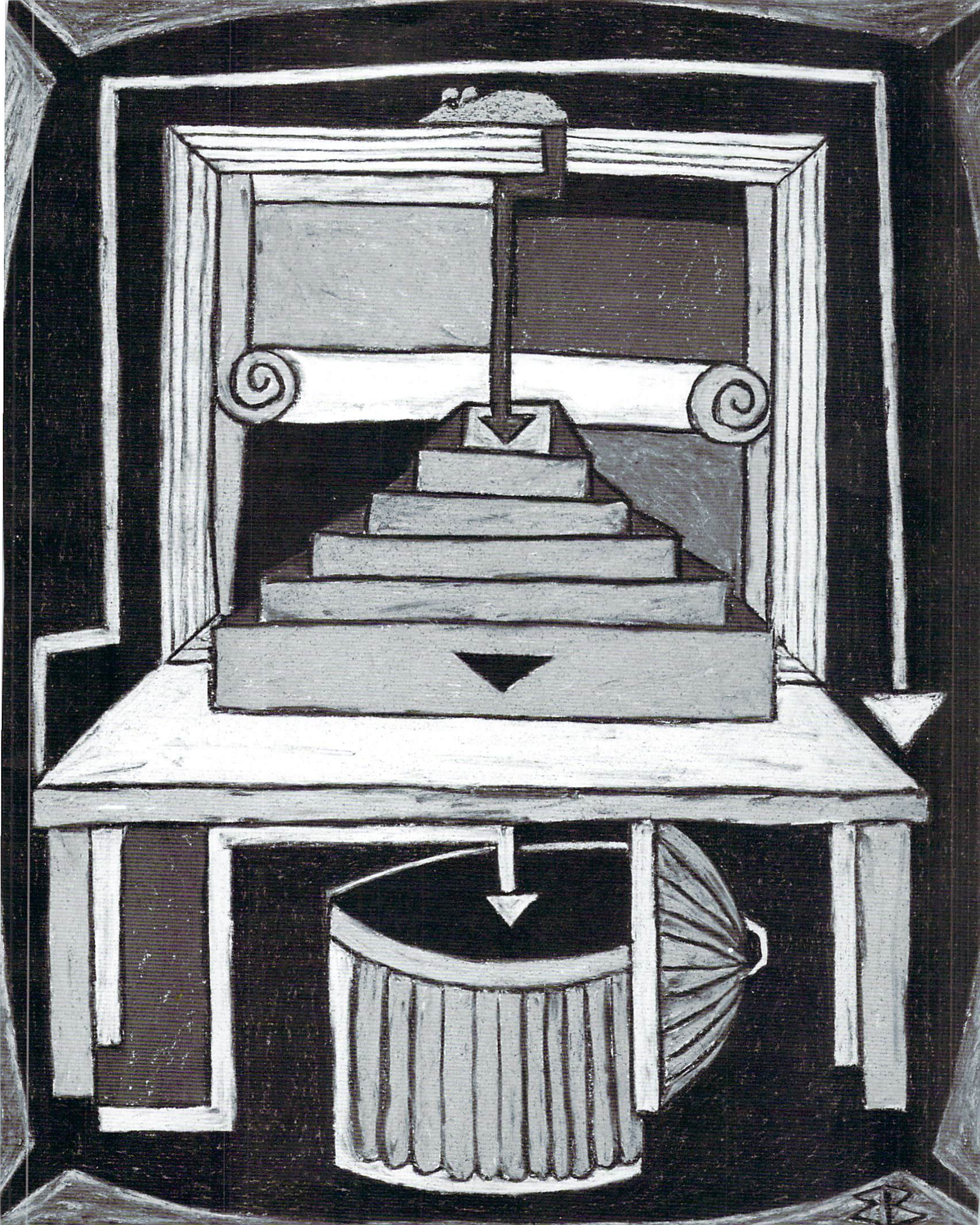
Tools: These are programs that perform some useful function; the Notepad, Icon Editor, Diskcopy program and Calculator are tools.

Getting Around in Workbench

Mouse, Icons, Windows, Drawers

The *right* mouse button is for making selections from menus; the *left* mouse button is for moving and activating things, such as moving, selecting or activating icons, or moving a cursor. Keyboard equivalents for these functions are as follows: left-ALT + left-Amiga key = left mouse button; right-ALT + right-Amiga key = right mouse button. To move the pointer by keyboard, use either Amiga key and the four arrow keys. Doing this while holding down Shift will cause the pointer to move a greater distance. ►

ILLUSTRATED BY ELIZABETH BRADY



Dressing Up Your Workbench: Using the Icon Editor

AREN'T YOU TIRED of clicking on those same little boxey drawers, when you could have light bulbs, rocket ships, ice cream cones or dinosaurs adorning your Workbench? Or how about little treasure chests that open when you click on them? The Icon Editor tool is a mini graphics program for redrawing all the icons that appear on the Workbench. It's easy to use, and a clever way to dress up and personalize your Workbench display.

The IconMerge program that comes on the 1.2 Extras disk is used along with the Icon Editor to produce icons that change shape or image when selected. (Two icons are actually "merged" into one, so that when you select the icon, the alter image appears; e.g., the IconMerge icon.) I suggest you move IconMerge to the System drawer with IconEd. The results to using these programs are strictly aesthetic, but it might make using Workbench a lot more fun!

Loading and saving icons are the only functions you are likely to have trouble with in the Icon Editor. This is because the requesters that you type into for the Load and Save options in the Disk menu require that you refer to icons in AmigaDOS form. (A little knowledge of CLI typing format will help a lot here.) It's not difficult if you remember where the icons are that you want to change, and what *type* they are.

Everything In Its Place

The five types of icons are: Disk, Drawer, Tool, Project and Garbage, which you probably have seen on the Icon Editor's confusing intro screen. Disk and Trashcan icons represent unique drawers and therefore are unique icons. Disk icons are always referred to as DISK, no matter what the disk's name is. Since in AmigaDOS the internal drive is DF0: and the external drive is DF1:, to load or save the internal disk's icon, you type DF0:DISK. Type DF1:DISK for loading or saving the external disk's icon. The Trashcan icon is referred to by name; of course this can vary depending on if you have renamed it. The Trashcan icon is, however, in its own category, and this is important when it comes to saving it, as described below.

Drawer icons are referred to by name. If the icon is, for example, the System drawer in the Workbench window in the internal drive, you would type DF0:SYSTEM. If it were an icon for a drawer you named TURKEYS on the disk in the external drive, you would type DF1:TURKEYS. If an icon is within a drawer that is within *another* drawer, such as the Notepad icon in the Utilities drawer in the Workbench drawer in the external drive, you must type DF1:UTILITIES/NOTEPAD. The slash tells the Amiga that the last typed name refers to an icon that is within the drawer named before the slash; no spaces are used—unless, of course, the icon name has spaces (a practice I don't recommend).

To confuse things, you can substitute a slash for DF0:, but not for DF1:. (An initial slash always refers to the system disk, or Workbench disk, which must be booted in the internal drive.) Therefore, you can type /SYSTEM/SAY to load or save the Say program icon in the System drawer on the Workbench disk in the internal drive. Just remember, an icon has to be referred to properly to be saved or loaded; the program won't load an icon if you do this wrong, but it *will* save things to places you don't want it to, and this can cause you a lot of grief.

From There and Back Again

Saving icons has a further condition, which points up the significance of recognizing icon types: Icons can only be saved from frames that had icons of *the same type* loaded into them. Strange but true. Therefore, a frame used to load a drawer icon can only be used for saving drawer icons; a frame used for loading tool icons (like the Notepad's) can only be used for saving tool icons. Project file icons (like textfile icons) must be saved from frames where Project files were loaded. The Trashcan can only be saved from a window it was loaded into; likewise for disk icons. This can become very confusing if you load icons into the nine available different frames and move them around, or erase, copy or merge them into other frames; especially since each time you enter an icon's name to load it, you erase the previously loaded icon's name from the requester. The one fail-safe method I have found for keeping this straight is to always save an icon from the *same* frame you first loaded it into. This is worth being careful about (write it down): Saving icons improperly can scramble things up in your Workbench; you will seriously regret it if you manage to separate an icon from its program, take my word for it! Again, always use a copy of an original disk.

One Plus One Equals One

The **IconMerge** tool that comes on the 1.2 Extras disk is used with the Icon Editor for creating icons that change metamorphically when you click on them. It actually merges two icons so that the second icon appears when the icon is selected. It is quite easy to use and the results are great fun.

Using IconMerge is simple if you remember the rules. Let's say you want to create a drawer icon that is a box that opens when you click on it. First, you load

the drawer icon and redraw it in its frame as the "before" icon—the closed box—and then save it to its proper place. Now you create the "after" icon, the open box. The most important things to remember about saving the "after" icon are that it must be saved in exactly the same sized box as the "before" icon, and you must name it something different than the "before" icon. You will have noticed that you have an option when saving icons: You can save them as entire frames or as boxed portions of the frame with the Frame and Save feature. If you (as I always do) box icons the size you want, you will have to make sure you save the "after" icon the same size. If you don't do this, the icon when selected will shift. An easy way to ensure this is to save things as whole frames, but this can be a waste of space. I usually put "register" marks in the frame to indicate the size frame I will save, being careful to put them outside of the area I want to save. This is easy if you put the marks on the "before" frame before saving it and then duplicate the frame and modify the copy for your "after" icon.

Once you have your "after" icon ready, save it to the same drawer (or type location) as its "before" icon by a name other than the name of the final merged icon; e.g., you could save it as TURKEYS.2 if the"before" icon is TURKEYS. Now leave the Icon Editor and activate IconMerge. Follow the simple directions: Type the name of the first icon, let's say TURKEYS, and hit return; type the name of the second icon, TURKEYS.2, return; then type the name of the final icon you want, TURKEYS. When you press return, the program will merge the icons. Now, close and reopen the drawer containing your merged icons (or, if it's a disk icon, reboot). You will find a loose copy of the "after" icon, which you can discard or store somewhere, if you want to use it again. If you change the merged icon with the Icon Editor, you will lose the "after" icon and will have to merge icons again; a good reason to store those loose icons away in a separate drawer. Many funny, descriptive and clever variations are possible with this feature. My Workbench contains doors that open, faces that change, a skinny cat that grows fat (the Expansion drawer) and other bizarre and humorous icons; I'm a confessed doodler at heart.

Icons Away

Although it takes some trial and error, the Icon Editor is one of the more creatively-oriented tools on the Workbench, especially when combined with IconMerge. Experiment, use your imagination, and you will be pleased with the results these sophisticated little tools have to offer.□

Icons represent disks and the drawers, programs and files on them. Double-clicking activates them and they can be dragged around, even between windows (excluding disk icons) representing different disks. Multiple icons can be dragged at the same time by holding down Shift while you select any number of icons (they will all appear highlighted), and without releasing the mouse button after selecting the last icon, dragging it; all the selected icons will move in a group.

Drawers have icons that when activated open windows. Drawers make it possible to group and store things of a similar nature together. Therefore, tools related to disks and the Amiga system come in the System drawer; the Notepad and Calculator—utility program tools—are in the Utilities drawer. When you place an icon within a window, you have actually placed the program that icon represents within the directory represented by that window. Drawers can be put within drawers within drawers, etc.; this is equivalent to directories with subdirectories with sub-subdirectories. Though it can provide insight into the way the Amiga handles things, you never need to understand the directory structure drawers represent to use them from Workbench. Although disk icons and the Trashcan icon are considered as representing drawers, they have some different properties, e.g., they cannot be placed within other drawers or deleted.

Windows in Workbench display what is in a drawer or provide the frames within which some programs run. You can easily manipulate a window by using that window's gadgets. To move a window, click the Left mouse button on the stripped bar at the window's top, and drag the window where you want it. Windows are sized by dragging the gadget at a window's bottom right. At the top right of windows are their back-front gadgets: By clicking on these, you place that window behind all others displayed on the screen at that time, or bring it to the front. This allows multiple windows to be open at once and displayed, as it were, in layers; you will find you often have to move and size windows to get at a particular window's front-back gadgets to bring it to the front. A gadget at the top left closes the window. Some windows have Scroll Bar gadgets, which allow you to visually scroll the contents of the drawer into view, vertically and horizontally. Disk windows have "fuel" gauges on the left side to tell you how full—or empty—the disk is.

When you click in a window, that window becomes active; only one window is active for input at a time, although numerous windows can be actively displaying program *output* simultaneously (multitasking). You will see this especially when a number of windows are opened and you need to move among them to work; when you move to a different window, you have to click within that window with the left mouse button first, before you can work in it. If a window has a menu bar associated with it, it will be displayed when that window is active and you press the right mouse button.

Moving Things

If you want to move something from one place to ►

another in Workbench, simply drag the thing to be moved to its new location. If you want to move a program from one drawer to another, you need not open the drawer's window you are moving it to; you can simply drag the icon over the drawer icon itself. If you move something from a drawer on one disk to a drawer on another disk, a copy will be made to place on the destination disk; the original will remain on the disk you dragged the icon from. Disks can be copied in this way also. (See below under Disk Operations.)

The Workbench Menu Bar

At the top of the Workbench screen is the menu bar. To make menu selections, you use the right mouse button and move the pointer to the menu title. When the menu opens, and with the button still held down, move the pointer to the desired selection, which will become highlighted; when you release the mouse button, the selection is made. If your menu selection opens a submenu, continue into that menu to the desired selection. If a menu selection appears in a *ghost* form (spotted and will not highlight), it indicates that the selection is unavailable for use with whatever you are currently doing. The menu bar itself displays the available free RAM memory (e.g., if it says 29016, you have about 29K of free RAM). You will notice this number rise and fall as you open and close things in the Workbench; it will give you a good indication of how much memory a particular thing uses.

A good way to make new empty drawers is to duplicate the Empty drawer and rename it . . .

The Workbench menu bar has three menus: *Workbench, Disk* and *Special.* Below are brief descriptions of all the options contained in these menus.

Workbench Menu Options

Open adds an optional way of opening windows, drawers and programs, instead of double-clicking on an icon. Simply select an icon and then choose Open. **Close** does the same as the close gadgets at the top-left of windows, except it only works for certain windows; try it to find out.

Duplicate is for duplicating disks, drawers, programs or files. (Duplicating disks is explained in the section below on Disk Operations.) To duplicate drawers, tools and files, simply select the appropriate icon and then select Duplicate. Drawers will be duplicated and, therefore, everything inside them too. A copy will be made and named "Copy 2 of. . .," or "Copy 3 of. . .," or whatever the number duplicate it is. A good way to make new empty drawers is to duplicate the Empty drawer and rename it whatever you want. **Rename** is for changing the names of disks, drawers and programs. To do this, select the icon of what you want to rename and choose Rename; a requester will appear into which you type the new name. When you have typed the new name, press return and the name with the icon will change. (You can't use a / (slash) or a colon in names since these are conventions used by the Amiga's operating system.) It is also advisable to avoid spaces in names, and, as you will find when icons get crowded, to avoid long names. Many people use conventions, such as suffixes like .txt or _Plist, to aid in easily differentiating things of different type, like text files and program listings.

Do *not* Rename the System drawer or the Expansion drawer. These drawers are *default* drawers for specific programs and changing their names will cause the Amiga to fail to locate some important programs. The Diskcopy and Initialize programs reside in the System drawer; the Amiga automatically looks there for them and they must stay there (don't move their icons from the System drawer window). Likewise, the expansion drawer is where "driver" programs are stored that are necessary for using hard-disk drives; the Amiga will look there for them (if you use a hard disk) and won't find them if the drawer is Renamed. If you Rename the Utilities drawer, you will have to change the default location of the Notepad (since the Amiga will look for it there).

Info will give you information about a disk, drawer or program if you choose it after selecting an icon. It will give you information including the type of object it is, its size or how full it is and whether it is a disk that is copy protected or changeable, or a program or drawer that is deletable. It is also used to change default directories for certain programs, such as the Notepad. **Discard** allows you to delete drawers, tools and files. Simply select the appropriate icon and choose Discard. You will be warned that you *can't get back* again what you discard.

Disk Menu Options

Empty Trash is used in conjunction with the Trashcan icon. It is another approach, besides Discard, to deleting things. When you drag an icon over the Trashcan icon or into its open window, you have essentially thrown the item into the garbage can, but you haven't "taken the trash out yet," i.e., it is still retrievable. You can open the Trashcan window and snatch out anything that's currently in there. If you want to discard the contents of the Trashcan, select the Trashcan icon and choose Empty Trash. You can discard a bunch of things at once this way. Remember, once you choose Empty Trash, the things in the can are forever gone—recycled back into available memory. **Initialize** must be used before you can store information on a new blank disk. Initializing a disk is described in the section below on Disk Operations.

Special Menu Options

Clean Up is used for organizing the contents of drawers. When a drawer is *first* opened, if you select Clean Up, all the icons in it will be put in nice neat rows. (Now you see why short names are preferrable.) Once your window is all in order, you can save the arrangement with **Snapshot**. Here's how: While holding the Shift key down, select *all* the icons you want to save in their positions. After they are all highlighted, go to the menu and choose Snapshot: After a little disk activity, they will stay the way you want them. Of course, you will often want to arrange icons separately—exactly where you want them—before taking a Snapshot,▶

instead of choosing Clean Up. If you want to save the location of a disk icon on the screen, select it and choose Snapshot. To save the *size* a disk window opens to as well as the organization of its inside icons, open the window, Clean Up or arrange the icons, select all the inside icons *and* the disk icon, then pick Snapshot. This also applies to drawer windows; if you select a drawer icon along with the icons in its open window, the window will henceforth open at that size.

Sometimes you will notice error numbers in Cancel/Retry requesters or the menu bar. **Last Error** will display, in the menu bar, the last error you made (on the Amiga, that is!), if you want to refresh your memory. **Redraw** will redraw the present screen if some Workbench program bug or failure has corrupted the screen display. (I have never had to use this feature; my crashes always require stronger medicine.) Finally, **Version** simply displays in the menu bar the versions of Kickstart and Workbench in use by the Amiga at that time.

Three swaps is a nuisance; 12 swaps is torture!

Disk Operations

Workbench gives you two ways to copy a disk. The first is in the *Workbench* menu on the menu bar and involves making a copy using one drive only. Select the icon of the disk that you want to copy and then choose **Duplicate**. (This can be done from an external drive, as well as the internal one.) Duplicate will copy the disk in stages, requiring you to swap the To and From disks. When you insert the From disk, the Amiga will "read" a section of the disk into memory and then ask you to place the disk to copy to in the drive. It then "writes" the copied section to the disk and prompts you to repeat the process—as many times as it takes, depending on the memory your Amiga has available. (With 512K, usually three swaps are required; with only 256K, as many as 12 swaps can be necessary if you have other things taking up memory space. Three swaps is a nuisance; 12 swaps is torture! No Amiga should be without at least 512K.) The Diskcopy icon in the System drawer, you will notice, simply tells you to go to this menu selection to make a copy.

The second method for copying involves two drives. This is done by dragging the icon of the disk you want to copy over the icon of a disk you want to copy onto. The disk you are copying to, of course, will be totally overwritten. The Amiga will tell you where to place the To and From disks. This is so easy to do, it's a great argument for having two drives.

New disks must be Initialized, or formatted, before you can store anything on them. This process also will erase the data on a disk and prepare it to receive new data. **Initialize** is in the *Disk* menu. To initialize a disk, select the disk and choose initialize. You will be warned that "all data will be erased," and if you select Continue, the disk will be formatted. Make sure you wait until this process is completed before taking a disk out. The newly formatted disk will be named "Empty"; use Rename to name it. It will also already have a Trashcan on it; Trashcans are put there automatically and cannot be deleted. If you select the Format icon in the System drawer it will tell you to go to this menu selection to do this. An Empty drawer from another disk can be moved onto the empty disk and duplicated for new places to store things.

One final note about disks: Never, never, *never* remove a disk or press the disk ejection button when the disk drive activity light is on. You can scramble the disk this way and make it unreadable; your only hope will be the Diskdoctor program accessed through the CLI (see the article on p. 16) or a third-party disk repair program. Always wait until the light is out and then wait some more. Or wait until the pointer on the screen returns to its normal form.

Workbench Tools

In addition to the functions performed by its menus, Workbench contains many tools that make it easier for you to work with your Amiga. The most important of these is **Preferences**. You use Preferences to configure your system. For instance, you can choose between a 60-column display and an 80-column one, tell the system what kind of printer you have and where it is attached, and even set the colors of your Workbench display. Other functions performed by Preferences include activating the CLI, configuring your serial port, and editing your Workbench pointer.

You can also use Preferences to set the internal clock built into your Amiga. You can then read the clock using the **Clock** tool. The Menu provided with the Clock lets you choose between a traditional clock and a digital one.

Other important tools are found in some of the standard Workbench drawers, specifically the System and Utilities drawers. In addition to the Format and Diskcopy tools mentioned earlier, the System drawer contains **Say**, a speech synthesis program; **SetMap**, a program that lets you change the language of your keyboard; a screen-printing program called **GraphicDump**, and the **Icon Editor** (see the sidebar on page 76). If you've activated the CLI from Preferences, the System drawer will also contain the CLI tool.

In the Utilities drawer you have the **Calculator** and the **Notepad**. The Notepad is especially interesting because it is in effect a miniature word processor. Using it, you can create and print documents that use multiple fonts and type styles. Notepad is one of the more interesting Workbench tools.

Closing Up Shop

The Workbench with its tools is a very handy place indeed to do your work or start your work from. The Workbench makes things easy to find and use and allows you to decide your own approach to your work. Putting the tools to practical and creative use will prove to you, over and over, the sophistication of the Amiga, the power and practicality of multitasking, and the difference in just doing work on a computer and doing work on an Amiga with style. The next time you've got a job to do, clear a space on your Workbench.■

Orders Only 1-800-221-6086
1-800-843-3485 AZ

Customer Support 1-602-258-4943

Computer Best

AMIGA™

GAMES

Alien Fires	$26
Archon	$26
Archon II Adept	$26
Arctic Fox	$26
Arena	$24
Autoduel	$32
Balance of Power	$32
Bard's Tale	$32
Bridge 4.0	$21
Champ Baseball	Call
Champ Basketball 2 on 2	$29
Champ GFL Football	Call
Champ Golf Volume 1	$36
Chessmaster 2000	$29
Deep Space	$32
Defender of the Crown	$32
Deja Vu	$32
Earl Weaver Baseball	$32
Faery Tail Adventure	$32
Flight Simulator II	$32
Gato	Call
Grand Slam Tennis	$32
Gunship	$29
Hacker	$29
Hacker II	$32
Halley Project	$29
Infocom Titles	$24-$32 ea.
Jet	$34
Keyboard Cadet	$26
King of Chicago	$32
Kings Quest I, II, III	$32
Leader Board	$26
L.B. Tournament #1	$15
Marble Madness	$32
Mastertype	$28
Mean 18	$26
Mean 18 Famous Courses	$13
Mindwalker	$35
O G R E	$26
One On One	$27
Phantasie	Call
Portal	$32
Return to Atlantis	Call
Rogue	$25
S D I	$32
Shanghai	$29
Silent Service	$26
Sinbad	$32
Skyfox	$26
Space Quest	$32
Starfleet I	$35
Super Huey	$26
The Pawn	$32
Ultima III	$39
Video Vegas	$22
Winter Games	$26
World Games	$26

EDUCATION

Algebra the Movie	$14
Discovery Math	$24
Discovery Spell	$24
Discovery Trivia	$24
First Shapes	$33
Geometry Motion Picture	$14
Great States	$28
Kid Talk	$35
Little Computer People	$32
Math Talk	$35
New Tech Coloring Book	$13
Perfect Score Sat	$51
Quizam	$23
Speller Bee	$35
Talking Coloring Book	$23
True Basic Math Titles	$36 ea.

POLICY: $3.00 minimum shipping. All prices subject to change (hopefully down!) No surcharge for credit cards. Send cashier check or money order for faster delivery. Personal checks take two weeks to clear. Defectives replaced with same item only - no exchanges or refunds. C.O.D., Foreign APO, School purchase orders accepted. AZ residents add 6.5% sales tax.

GRAPHICS

3-D Animations	$14
Aegis ArtPack #1	$22
Aegis Draw	$80
Aegis Draw Plus	$170
Aegis Impact	$63
Animator/Images	$99
Car Tips For Women	$14
Christmas Video	$14
Computer Art Gallery	$14
Deluxe Companion	$20
Deluxe Help	$13
Deluxe Paint 2.0	$85
Deluxe Paint Art Parts #2	$19
Deluxe Print	$65
Deluxe Video	$65
Digi Paint	Call
Digi View	$143
DPaint Art & Utility Disk	$19
DPrint Art & Utility	$19
Dynamic Cad	Call
Gallery of Images	$14
JDK Fonts	$22 ea.
Print Master Art Gallery I	$19
Print Master Art Gallery II	$19
Print Master Plus	$32
Pro Video CGI	$131
Zuma Fonts 1, 2, 3	$23 ea.

PRODUCTIVITY

2 + 2 Home Management	$64
Analyze Vs 2.0	$97
BBS PC	$62
BTS	Call
CLI Mate	$26
D Buddy	Call
DataMat	Call
Dbman	$106
Financial Cookbook	$33
Financial Plus	$247
Flow	$74
Gizomz	$37
Goldspell	$29
HomePack	Call
Isgur	Call
MaxiComm	$32
MaxiDesk	$45
MaxiPlan	$97
MaxiPlan Plus	$130
Mi Amiga File II	$61
Mi Amiga Ledger	$61
Mi Amiga Word	Call
On Line	$46
Organize	$63
Pagesetter	$110
Paperclip Elite	Call
Phasar	$62
Pro Write	Call
Publisher 1000	Call
Rags to Riches/GL	$139
Scribble	$63
Superbase	$95
Talker	$47
Unicalc	$61
Vip Professional	$153
Zing!	$49

SOUND & MUSIC

Deluxe Music	$65
Future Sound	$129
Gallery of Holiday Music	$14
Gallery of Songs Hldy Ed	$14
Instant Music	$32
It's Only Rock & Roll	$20
Music Studio	$30
Perfect Sound	$59
Sonix	Call

WE ACCEPT

VISA · MasterCard · AMERICAN EXPRESS Cards Welcome

HARDWARE

256k Ram	$78
A-Time	$49
C-64 Emulator	Call
Camera W/lens WV1410	$225
Easy L	$395
Midi Device	$38
Midi Gold	$64
Modem/online/cable	$199
Mousepad (large)	$7
Mousepad (small)	$6
Penmouse Plus	$240
Starboard 2MB	Call
Starboard Multifunction	Call

PROGRAMMING

Digital Link	$55
Diskwik	$36
Dos-2-Dos	$41
Grabbit	$23
Lattice "C" Compiler V3.1	$165
Lisp	$139
Macro Assembler	$70
Manx "C" (Comm) V3.4	$330
Manx "C" (Dev) V3.4	$203
Marauder II	$29
Metascope Debugger	Call
Mirror	$34
Mirror Hacker Package	$34
Modula II Commercial	$220
Modula II Developer's	$112
Modula II Personal	$65
Pascal	$70
Shell	$49
Text - Ed	$23
Toolkit	$35
True Basic	$96

Orders To:

Computer Best
P.O. Box 48407
Phoenix, AZ 85075

Play Ball!

In your FREE EA T-Shirt!

EA

Buy any two EA programs (excluding data disks) and receive a T-shirt direct from Computer Best. This offer ends real soon! 4-1-87 to 6-30-87

Executive Control: Introduction to the Amiga's Kernel

Behind the Amiga's multitasking software, flexibility and ease is a group of complex and powerful low-level routines—the Exec. This article is about learning to use the Exec's functions.

By David T. McClellan

AmigaDOS is one of the most powerful microcomputer operating systems outside of Unix. And unlike Unix, it runs well on floppies. AmigaDOS is built on top of both sophisticated hardware and a workhorse of a kernel, known as the Exec. (These are the core system-software routines that control the major functions of the computer.) The user-callable AmigaDOS functions, the CLI and Intuition are built on top of Exec functions (file input/output, for example, uses the Exec trackdisk device handler). In this article, I cover mainly the Exec functions, rather than higher-level AmigaDOS routines; I will discuss these functions from the point of data structures and routines that the Exec provides, and will show how you can use them.

AmigaDOS is a multitasking operating system, running on top of the multitasking Exec kernel. This complicates the Exec's job enormously over what a single-task operating system, such as MS-DOS, has to worry about. At any time, several programs may be contending for the same resources: CPU time, devices, memory and user-interaction. The Exec has to manage these demands without letting them step all over each other.

To create a separate task at the CLI level, you use the Run command. When this is done, the program starts and the CLI returns for another command while the program is still running; it does not wait until the program terminates. A user program in C or another language can do this as well, although it takes a little more work. In the following discussion, I'll describe things from the point of view of C; all the data structures I'll use are C structs or struct pointers. First, let me describe tasks and how they differ from processes.

Tasks and Processes

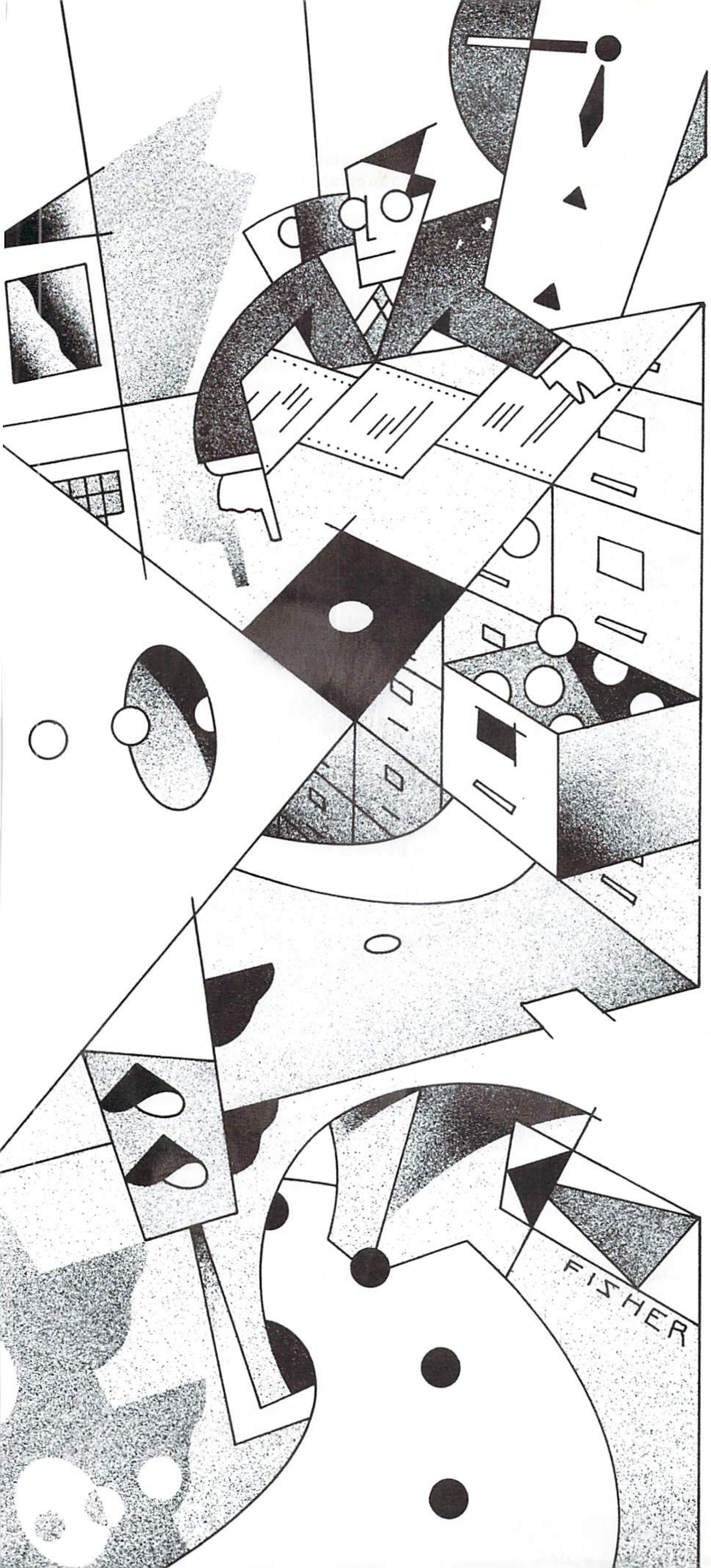
To the Exec, a task is a piece of code in memory, with a known entry point, its own stack and a task control block describing it linked into the kernel's list of tasks (more on lists in a bit). It can be alone, or part of a single loaded program that sets up parts of itself (subroutines) as separate, independently executing tasks. However, tasks that are part of another program can only call on Exec-level functions; only the main program should call AmigaDOS-level functions and do I/O with AmigaDOS devices.

A process is a task with extras: a memory space and process context separate from any other running program. It is typically loaded by another program (such as the CLI) and started up as a process with AmigaDOS functions. A process can contain several tasks. What the Run command creates is actually a separate process. AmigaDOS handles processes and their requests; the Exec handles tasks.

To create and start up a task, a program must allocate and initialize a task struct (described in the *exec/tasks.h* include file), allocate stack space for the task, and call the Exec routine AddTask() with the task struct, the address of the entry point of the code, and the address of a termination routine for the task if one exists (this can be Null). This call will link the new task in with the Exec's current list of tasks, and start up the code. When done, the new task will terminate and delete itself; the program that started it must free the stack space and the task struct. The new task and its parent can communicate via signals and message passing, which I'll discuss later.

Besides its own stack and entry point, each task has a priority, and may have a name. The Exec uses a task's priority to put the task in the proper place on the task list, to schedule how much Amiga time it gets in relation to other tasks. The Exec also keeps a variety of other lists, and so has a generalized list structure and routines to manage lists for the kernel and user pro-

ILLUSTRATED BY MARK FISHER



grams. Each item on such a list is called a Node. The following example shows the Node struct as defined in the include file *exec/nodes.h*:

```
struct Node {
struct Node *ln_Succ; /* Ptr to next Node in list */
struct Node *ln_Pred; /* Ptr to previous Node */
UBYTE  ln_Type; /* Type of Node; the Exec */
        /*  supports several and */
        /*  you can define more. */
BYTE ln_Pri; /* Priority, -128 to 127 */
char  *ln_Name; /* Name of Node, can be  */
       /* NULL.  */
};
```

Every list node has the above standard node header and an arbitrarily-sized area tacked on after it for various kinds of data. What comes after the header is up to the Exec and you; for the task list, it is the rest of the task structure. Tasks and processes can do more when they can communicate with each other. Inter-task communication allows the programmer to separate independent parts of an application, which can be run alone, reducing overhead in the main application. Examples are device "watchdogs," which alert the main application program when device events occur, and server tasks, which perform common functions for several user programs. A program using a server makes a request and then goes on about its business while the server does the request. Multiuser database systems like to do this to gate access to the database. The program in Listing 1, *tlogger.c*, is an example of a server task.

Signals, Messages and Ports

Tasks can communicate in one of two ways: simple status information via the task "signal" mechanism, and more complicated communications containing arbitrary data with messages. Each task is allowed up to 32 signals, 16 of which are usually used by the system, the others available for users. A program wishing to allocate a signal calls AllocSignal(), requesting a specific signal number (which it may or may not be able to get), or more usually requesting any available signal by calling AllocSignal() with −1. The program can then let other tasks or the Exec know what the signal bit is, make requests and wait for receipt of the signal, via Wait(), for the requests to be done. The single argument to Wait() is the signal's bit number; Wait() can wait for several signals joined together in its argument, as in:

```
Wait (phone_sig | toaster_sig | doorbell_sig);
```

To allow tasks to pass back and forth more information than the simple yes/no of a signal, the Amiga Exec provides "message passing," a mechanism whereby a program sets up a message struct, tells the kernel to pass that struct to another program waiting for it, and waits for a reply. The receiving program doesn't have to be concerned with where the message came from, only its contents; the sender does not have to worry about how to alert the receiver (Exec will do that). The mechanisms are simple; they provide the basis of communication for the three small programs I wrote to demonstrate things in this article. ►

A program wishing to receive messages must set up a message "port," which it names with a name known to programs that will send it messages. The CreatePort() function allocates and initializes a port, and returns a pointer to the port for the caller to use with WaitPort() when waiting for messages. Message ports are kept on Exec lists, so CreatePort() takes a name and a priority as arguments, allocates memory for the port, fills in the node header with its arguments and adds the port to the appropriate list (public or private, depending on whether the name is non-null). Senders get the Exec address for a public port by calling FindPort() with the name of the port and getting an address in return. Senders set up message ports for the receiver's reply, similar to having a telephone at each end of a phone conversation. The reply port can be public or private; in my examples they are private.

A message is a three-part data structure, the first two parts of which are common across all messages. The first part is a list Node header; the second part contains the reply port address for the message and the length of the third part. The third part of the message can be any data structure desired, so long as senders and receivers agree on it. See the TlogMsg struct in file *tlogger.h* in Listing 4 for an example.

To the Exec, a device is a set of well-defined operations that provide access to some physical object (disk, timer, gameport, etc.).

The basic sequence of calls for a receiver is:

```
myport = CreatePort (name, priority);
...
WaitPort (myport);
nextmsg = GetMsg (myport);
  /* Process the message ... then */
ReplyMsg (nextmsg);
  /* Go back and do it again... */
  /* When done receiving messages, do */
DeletePort (myport);
```

The sender, on the other hand, does this:

```
  /* Fill in the message ... then */
PutMsg (mymsg);
WaitPort (replyport);
  /* Now go on and do other things ... */
```

Input and Output

Message ports are also used when doing I/O with Exec. A program that wants to do Exec-level I/O must set up a message port for the I/O device to reply to when a requested I/O action is complete; this is illustrated in file *joymsg.c* in Listing 2. To the Exec, a device is a set of well-defined operations that provide access to some physical object (disk, timer, gameport, etc.). Each physical device has an associated driver that provides the standard operations as well as any extra device-specific functions required. A device driver may support more than one instance, or "unit," of the device. For example, the gameport has two units—the left and right mouse/joystick ports. The Amiga has eleven standard devices: the timer, the trackdisk (floppy), keyboard, gameport, input (keyboard and gameport combined), console (input and screen output), audio, narrator (speech synthesizer), serial, parallel and the printer device. A device can be a combination of other devices, or be based on another device.

To interact with a device, a program must first set up a message port, initialize an IOStdReq struct with CreateStdIO(), and open the device with OpenDevice. Exec devices are named, so to open the gameport device for I/O a program would do:

```
ioport = CreatePort(NULL, 0);
iomsg = CreateStdIO (ioport);
error = OpenDevice ("gameport.device", unit_number,
        iomsg, 0);
            /* If error is non-zero, the device
            could not be opened for some reason. */
```

After opening an Exec device, the IOStdReq struct is used to communicate with the device. The IOStdReq struct is essentially an extended message struct—the first part is a standard message structure, followed by fields used by the I/O subsystem. This allows the I/O subsystem to use message passing to communicate with your program. According to *exec/io.h*, it is:

```
struct IOStdReq {
     struct Message io_Message;
     struct Device  *io_Device;
     struct Unit *io_Unit;
     UWORD io_Command;
     UBYTE io_Flags;
     BYTE  io_Error;
     ULONG io_Actual;
     ULONG io_Length;
     APTR  io_Data;
     ULONG io_Offset;
};
```

The message part and device/unit pointers are set by Exec when you call OpenDevice(). The io_Command field is used to tell the device which operation to perform when you send it a request with either DoIO(), SendIO() or BeginIO(). The standard I/O commands supported by every device are: CMD_RESET (initialize), CMD_READ, CMD_WRITE, CMD_UPDATE (force internal buffers to be sent), CMD_CLEAR (clear buffers instead of sending), CMD_STOP (halt device), CMD_START (restart stopped device, like a printer) and CMD_FLUSH (abort all pending I/O requests). I/O requests are queued to the device—in a multitasking system this is an absolute necessity—so at any one time several may be awaiting service by the device handler. A user program can do its I/O in one of two ways: synchronously, by queueing an I/O request with DoIO(), which will not return until the request is done, or asynchronously with SendIO(), which will queue the request and return immediately. When using SendIO(), the user program sends out the request, goes about its business, and either checks on the request's status with CheckIO() from time to time or eventually gets to a point where it can wait on the request to finish and then calls WaitIO(). BeginIO() operates at a lower level than DoIO() or SendIO(), requiring more knowledge of the device driver to use. Its I/O may be either synchronous or asynchronous, depending on the device.

The standard I/O routines (see "Arrivals and Departures: Input, Output and C," page 67) used in C programming, such as *fopen(), printf(), fputs(),* and so on, are implemented on top of AmigaDOS file and console I/O routines, which in turn are built upon Exec functions.

My tutorial programs use a mixture of the C functions (for file I/O and communicating error messages) and Exec functions for demonstration purposes and for getting at things C *stdio* functions can't do. Used with care, the three different layers of I/O can all be called within the same program.

Interrupts

Tasks and device handlers must be able to be interrupted when certain events happen, such as when a physical device has a piece of data that may be lost if not grabbed quickly, or when a hardware timer goes off. These are like signals in that they communicate only status information; they are unlike signals in that a program does not Wait() for them, but gets interrupted when they happen. There are both hardware interrupts, from physical devices, and software interrupts, which programs can send.

When an interrupt occurs, it must be caught and serviced as quickly as possible; for example, if the serial port says a byte is ready, the serial device handler must grab the byte and store it in a memory buffer or the next byte coming along will wipe it out. Interrupts are prioritized so that very important ones will be serviced before less important ones; the handling of a low-priority interrupt can be interrupted by the handler of a higher-priority interrupt. When an interrupt server is done, the interrupted task(s) continue where they left off. A user program can take over handling a given interrupt if it has to grab data in the fastest way possible. To do this, the program must set up an interrupt struct with the interrupt number and a pointer to a routine that will service the interrupt, and call AddIntServer() to add the handler to Exec's set of interrupt handlers. The routine will be called whenever its interrupt occurs. When the program as a whole is done, it must call RemIntServer() to tell Exec to take over handling that interrupt again.

Memory Allocation

All these services require memory for data structures, allocated as needed. Programs ask Exec to allocate and return memory on an "as needed" basis, using AllocMem() and FreeMem(). When you need a chunk of memory, call AllocMem() with the amount needed and it will return a pointer to the memory; to free it later, call FreeMem() with the pointer and the size of the memory chunk. For example, in the tutorial programs I allocate memory for my personal message struct (struct TlogMsg in file *tlogger.h*) and later free it with:

```
struct TlogMsg *out_msg;
     ...
     out_msg = (struct TlogMsg *)
               AllocMem (sizeof(struct TlogMsg),
               MEMF_PUBLIC | MEMF_CLEAR);
     ... /* Use out_msg to send messages */ ...
     FreeMem (out_msg, sizeof(struct TlogMsg));
```

If AllocMem() cannot find a piece of memory big enough to fill your request, it will return NULL, so always check the pointer you get back.

AllocMem() has other features, performed according to its second argument (the MEMF_PUBLIC | MEMF_CLEAR in the example). It can allocate "public" memory or "chip" memory (the first 512K) on request. The graphics and audio chips require data structures they use, such as bitmaps or screens, to be in chip memory; AllocMem() can guarantee this placement. Exec will take memory anywhere in public space, thus my use of the MEMF_PUBLIC bit. AllocMem() will also zero the allocated memory if it is passed the MEMF_CLEAR bit.

Conclusion

That's a brief overview of the Exec kernel, a sophisticated and powerful chunk of software. The one facility I have not covered directly is use of the in-memory libraries for graphics, audio, and so on; they have been demonstrated (and will continue to be demonstrated) in other tutorials in *AmigaWorld*, and their use is explained in detail in the Rom Kernel Manual. The tutorial programs contained in the accompanying listing demonstrate message passing, device I/O and memory allocation, and indirectly show multitasking. The receiver/server program is *tlogger.c*; it operates by allocating a public message port, opening a file and then waiting for messages, and copying them to a file until one of the messages is the word "quit." It then drains off any remaining messages and exits.

The two sender programs, *keymsg.c* (Listing 3) and *joymsg.c* (Listing 2), interact with the user to get messages to send to *tlogger* (Listing 1). Program *joymsg* waits for button presses on a joystick attached to the right (rear) gameport; it will send a message to *tlogger* for each one it gets, and quit after five button presses/releases. *Keymsg* uses C's *stdio* routines to get text from the keyboard, which it sends to *tlogger*. It quits when the user enters the word "quit," after first sending that message to *tlogger*. Include file *tlogger.h* contains the common message struct used by all three programs, and various shared #defines. It also contains the instructions for running the programs and checking the results.

Modify the programs and use them as you want. The Exec is full of functionality; be prepared to make a few mistakes (I did!), and have fun with it. □

Address all author correspondence to David C. McClellan, 104 Chevron Circle, Cary, NC 27511.

***Listing 1.** tlogger.c*

```
          /*-----------------------------------------+
           |              tlogger.c                  |
           | Logs messages addressed to port named in|
           | tlogger.h to file named in tlogger.h.  Used
           | in AmigaWorld Exec functions Tutorial.  |
           | D.T.McClellan for AmigaWorld. Jan. 87   |
           | See tlogger.h for running instructions. |
           +-----------------------------------------*/

#include <stdio.h>
#include "tlogger.h"
                    /* This defines the data structures used by */
                    /* the three programs, and incidentally     */
                    /* includes the needed exec .h files.       */
```

Listing continued on p. 86

```
#define TLOG_INIT_MSG"TLOGGER: starting up\n"

struct MsgPort *in_port;       /* tlogger's port */
struct TlogMsg *in_msg;
              /* Most recently received message */

char local_buf[MAX_MSG_TEXT+1];

      /* Local buffer to copy message text into */
      /*   before replying to message and free   */
      /*   the caller's message struct.          */

main ()
{
     FILE *logfile;           /* Log file fileptr */
     UWORD msglen;
            /* Length of text in current message */


/*----------------------+
| Create and Install Port |
+-----------------------*/

   /* Allocate signal and memory for port */

   if ((in_port = CreatePort (TLOG_PORT_NAME, 0))
        == NULL)

   {      /* No room for port */
    printf ("tlogger: Not enough memory
            to allocate port.\n");
           exit (-1);
   }
```

```
/*--------------+
| Open Log File|
+--------------*/

  if ((logfile = fopen (TLOG_FILE_NAME, "w"))
       == NULL)
  {
   DeletePort (in_port);
   printf ("tlogger: Cannot open Log File %s\n",
           TLOG_FILE_NAME);
   exit (-2);
  }

  fputs(TLOG_INIT_MSG, logfile);
                /* Put out my init msg */


/*-----------------------------------------+
| Loop: Wait for, Reply to, and Log Messages |
+------------------------------------------*/

  for (;;)      /* Eternal Loop */
  {
     (void) WaitPort (in_port);

               /* Wait for a message to    */
               /*   come along.            */
               /* It returns a ptr to the  */
               /*   message, but we must   */
               /*   use GetMsg to actually */
               /*   dequeue the message.   */

        while ((in_msg = GetMsg (in_port)) != NULL
          {
     /* Log the message, reply, check
                for ending message */

        msglen = in_msg->msghead.mn_Length;
        strncpy (local_buf, in_msg->text, msglen);

        ReplyMsg (in_msg);
   /*   access to its message struct anymore.   */

        fputs(local_buf, logfile);
        if (local_buf[msglen-2] != '\n')
            /* Make sure each log message has a */
            /* newline terminator.              */
           fputc ('\n', logfile);

        if (! strncmp (local_buf, TLOG_QUIT_MSG,
            TLOG_QUIT_LEN))                /* Done *
           goto done_logging;
      } /* end while GetMsg */

   } /* end for (;;) */


   /*-----------------------------------+
   | Close down Port and Close Log File |
   +----------------------------------*/

   done_logging:
   /* First, drain the port of any later
                waiting messages */
      while ((in_msg = GetMsg (in_port)) != NULL)
         ReplyMsg (in_msg);

   /* Now free the port and allied stuff,
                and close the file */
         DeletePort (in_port);
         fclose (logfile);
         exit (0);
   }
```

Listing 2 follows on p. 88.

Listing 2. joymsg.c

```
/*-------------------------------------+
|              joymsg.c                |
| Send joystick i/o event messages to  |
| tlogger.  Each button press and re-  |
| lease and timeout is sent as a log   |
| message to tlogger.                  |
| D.T.McClellan for AmigaWorld. Jan. 87|
| See tlogger.h for running instructions.|
+-------------------------------------*/

#include "tlogger.h"
#include <devices/gameport.h>
#include <devices/inputevent.h>

/*--------+
| Defines |
+--------*/

/* Joystick i/o timeout - 30 second without a */
/* trigger (30 seconds * 60 ticks per second) */
#define JOY_TIMEOUT     30 * 60

/* Message text and lengths.  Lengths include
   trailing \0 which C automatically adds.   */

/* Initialization message */
#define JOY_INIT_MSG    "JOYMSG: starting up\n"
#define JOY_INIT_LEN    21

/* Termination message */
#define JOY_QUIT_MSG  "JOYMSG: done; see you later\n"
#define JOY_QUIT_LEN  33

/* Button down message */
#define BUT_DN_MSG    "JOYMSG: button DOWN (closed)\n"
#define BUT_DN_LEN    30

/* Button up message */
#define BUT_UP_MSG    "JOYMSG: button UP (open)\n"
#define BUT_UP_LEN    26

/* Timeout message */
#define TIMEOUT_MSG   "JOYMSG: button timeout\n"
#define TIMEOUT_LEN   24

/*-------------------+
| Global Data Structs |
+-------------------*/

struct InputEvent *joy_io_buff;
        /* Buffer for i/o event data */
struct IOStdReq *joy_io_msg;
        /* I/O Message */
struct MsgPort *joy_io_port;
        /* Gameport i/o rendezvous port */
struct MsgPort *out_port;
        /* tlogger's port */
struct MsgPort *reply_port;
        /* Msg reply port */
struct TlogMsg *out_msg;
        /* Message to tlogger */

/*---------------------------------------------+
| Close Up Shop Flags and Container.  These flags |
| are set in close_flags by the initialization |
| code, and used by the close_up_shop routine to |
| determine what needs to be done on termination |
| of joymsg.                                   |
+---------------------------------------------*/
```

Listing continued on p. 90

$4.95 For A Monthly Disk Filled With High-Quality Programs.

Introducing...

SOFTWARE DIGEST

your door to the world of high-quality public domain software and shareware

When you order *Software Digest*, **each month you will receive a floppy disk filled with the best and newest public domain programs and shareware**, conveniently delivered to your door. At *Software Digest*, our staff continually evaluates hundreds of public domain and shareware programs. We then compile the best, and organize these quality programs for nationwide circulation via the monthly disks you will receive. All software we distribute are the winners of winners and have proved themselves to be the most useful or entertaining.

Software Digest distributes almost every type of program you will ever need. The coming disks of *Software Digest* will contain programs such as **TEXT PROCESSING PACKAGES, SPELLING CHECKERS, SPREADSHEETS, DATABASES, TERMINAL EMULATORS, PLOTTING SOFTWARE** (some can generate 3D graphics), **PROGRAMMING LANGUAGES** (C, Lisp, Forth, and more), **UTILITIES, PRODUCTIVITY TOOLS, SOPHISTICATED DRAWING PACKAGES, ARTWORK, ARCADE GAMES** (Some are 3-dimensional; 3D glasses required.), **BOARD GAMES** (such as Othello, Yahtzee, etc.), **FUN PROGRAMS, AND JUST ABOUT ANYTHING YOU CAN THINK OF.** You will receive all these programs, plus any other quality programs that are available in the future, through *Software Digest*'s monthly disks.

Whenever significantly-improved versions of programs are released, *Software Digest* will deliver them to you as soon as possible to keep you up to date. The quality we bring you will definitely surpass your expectations!

Best of all, the subscription fee is only slightly higher than the cost of the blank disks alone. Why so little? Because your payment covers only the service costs: disks, duplication, processing, and packaging.

You have already invested a great deal in your Amiga. Why not get the most out of it? Just a few dollars per month will bring you so much convenience, productivity, opportunity, and fun!

Remember: the earlier you start your subscription, the sooner you will receive all of these benefits.

These programs are intended to be available for all Amiga owners to use. Here is a totally new dimension of quality software awaiting you, and it's available at a very low cost (only a few dollars per month). Why not subscribe today?

Attention, shareware developers!
If you have developed any high-quality programs and would like *Software Digest* to help you distribute them as shareware, please contact ***Software Digest Editorial Offices, MIT Branch P.O. Box 315, Cambridge, MA 02139.*** Any other submissions—such as public domain software, artworks, articles, etc.—are welcomed as well. Please submit them to the address above.

Software Digest Subscription Form

YES, I would like to order...

☐ **The current issue of *Software Digest***
$8.95 plus $1.00 for postage and handling.

☐ **A half-year subscription, 6 issues *($6.95 per issue)***
$41.70 plus $6.00 for postage and handling.

☐ **A full-year subscription, 12 issues *($4.95 per issue)***
$59.40 plus $12.00 for postage and handling.

Please print.

Name ______________________

Address ______________________

City ______________ **State** ________ **Zip** ________

Please make check payable to Software Digest and mail it, along with this subscription form, to Software Digest, Subscription Service, MIT Branch P.O. Box 315, Cambridge, MA 02139.
Please allow four to six weeks for delivery.

Ordering form may be photocopied

from p. 88

```
#define CL_REP_PORT             0x0001
#define CL_JOY_PORT             0x0002
#define CL_OUT_MEM              0x0004
#define CL_JOY_STDIO            0x0008
#define CL_JOY_DEVICE           0x0010
#define CL_GAME_TYPE            0x0020
#define CL_JOY_BUF              0x0040

unsigned long close_flags = 0;
              /* Flagword used by initialization */
              /* and close_up_shop routine.      */

/*----------------+
| Called Routines |
+----------------*/

extern struct IOStdReq *CreateStdIO();

main()
{
  struct GamePortTrigger button;
  int button_count;

/*-----------------------------------------+
| Open message ports; setup message struct |
+-----------------------------------------*/

  if ((out_port = FindPort (TLOG_PORT_NAME)) == NULL)
  {
    printf ("joymsg: Cannot find tlogger's port.\n");
    exit (-1);
  }

  if ((reply_port = CreatePort (NULL, 0)) == NULL)
  {
    printf ("joymsg: Cannot allocate reply port\n");
    close_up_shop();
    exit (-2);
  }
  close_flags |= CL_REP_PORT;

        /* Allocate and init message struct */
  if ((out_msg = (struct TlogMsg *) AllocMem (sizeof
     (struct TlogMsg), MEMF_CLEAR | MEMF_PUBLIC))
      == NULL)
  {
    printf ("joymsg: Insufficient memory
             for my message.\n");
    close_up_shop ();
    exit (-3);
  }
  close_flags |= CL_OUT_MEM;

        /* Initialize message invariants */
  out_msg->msghead.mn_Node.ln_Type = NT_MESSAGE;
  out_msg->msghead.mn_ReplyPort = reply_port;


/*--------------------------------------------+
| Setup i/o data structs, open gameport device |
+--------------------------------------------*/

  if ((joy_io_port = CreatePort (NULL, 0)) == NULL)
  {
    printf("joymsg: Could not create
            message port for i/o\n");
    close_up_shop ();
    exit (-4);
  }
  close_flags |= CL_JOY_PORT;

  if ((joy_io_msg = CreateStdIO(joy_io_port)) == NULL)
  {  /* Setup i/o request block to talk to gameport */
    printf("joymsg: Could not set up
```

```
            i/o request block\n");
    close_up_shop ();
    exit(-5);
  }
  close_flags |= CL_JOY_STDIO;

  if ((OpenDevice("gameport.device", 1,
                  joy_io_msg, 0)) != 0)
  {   /* Open right game port (number 1) */
    printf("joymsg: Could not open
            gameport device\n");
    close_up_shop ();
    exit(-6);
  }
  close_flags |= CL_JOY_DEVICE;

  if (joy_io_type (GPCT_ABSJOYSTICK) != 0)
      /* Set gameport device type to joystick    */
  {
    printf("joymsg: Could not set type of right
            gameport\n");
    close_up_shop ();
    exit (-7);
  }
  close_flags |= CL_GAME_TYPE;

        /* Set the button info */
  joy_io_msg->io_Command = GPD_SETTRIGGER;
  joy_io_msg->io_Length =
               sizeof(struct GamePortTrigger);
  joy_io_msg->io_Data = (APTR) &button;
  button.gpt_Keys = GPTF_UPKEYS + GPTF_DOWNKEYS;
  button.gpt_Timeout = JOY_TIMEOUT;
  button.gpt_XDelta = 100;     /* Large moves */
  button.gpt_YDelta = 100;

  if (DoIO(joy_io_msg) != 0)
  {
    printf("joymsg: Couldn't set
          gameport trigger type\n");
    close_up_shop ();
    exit (-8);
  }

  if ((joy_io_buff = (struct InputEvent *)
       AllocMem (sizeof (struct InputEvent),
       MEMF_CLEAR | MEMF_PUBLIC)) == NULL)
  {
    printf ("joymsg: Couldn't allocate
            input buffer\n");
    close_up_shop ();
    exit (-9);
  }

  close_flags |= CL_JOY_BUF;

/*--------------------------+
| Initial message to tlogger |
+--------------------------*/

  strcpy (out_msg->text, JOY_INIT_MSG);
  out_msg->msghead.mn_Length = JOY_INIT_LEN;
  PutMsg (out_port, out_msg);
  (void) WaitPort (reply_port);

/*--------------------------------------------+
 Loop, waiting for I/O events and sending messages
 on them to tlogger until button is hit 5 times.
+--------------------------------------------*/

  button_count = 0;
  for (;;)
  {
/* Get one i/o event each Sendio to the gameport */
/* We are ignoring joystick movement, available  */
/* by looking at joy_io_buff->ie_X and ie_Y;     */
/* we just catch button events and timeouts.     */
     joy_io_msg->io_Command = GPD_READEVENT;
```

```
      joy_io_msg->io_Data = (APTR) joy_io_buff;
      joy_io_msg->io_Length =
                  sizeof(struct InputEvent);
      joy_io_msg->io_Flags = 0;

      SendIO (joy_io_msg);
      (void) WaitPort (joy_io_port);
      (void) GetMsg (joy_io_port);

      if (joy_io_buff->ie_Code == IECODE_LBUTTON)
      {
        strcpy (out_msg->text, BUT_DN_MSG);
        out_msg->msghead.mn_Length = BUT_DN_LEN;
      }
      else if(joy_io_buff->ie_Code ==
              (IECODE_LBUTTON + IECODE_UP_PREFIX))
      {
        strcpy (out_msg->text, BUT_UP_MSG);
        out_msg->msghead.mn_Length = BUT_UP_LEN;
        button_count++;
      }
      else      /* Time out */
      {
        strcpy (out_msg->text, TIMEOUT_MSG);
        out_msg->msghead.mn_Length = TIMEOUT_LEN;
      }

      PutMsg (out_port, out_msg);
      (void) WaitPort (reply_port);

      if (button_count >= 5)
        break;
  }
/*----------------------------+
| Send final message and quit |
+----------------------------*/

      strcpy (out_msg->text, JOY_QUIT_MSG);
      out_msg->msghead.mn_Length = JOY_QUIT_LEN;

      PutMsg (out_port, out_msg);
      (void) WaitPort (reply_port);

      close_up_shop ();
      exit (0);
  }
  close_up_shop ()
      /* Close up and deallocate all things */
      /* allocated/opened. */
  {
      if (close_flags & CL_GAME_TYPE)
         (void) joy_io_type (GPCT_NOCONTROLLER);

      if (close_flags & CL_JOY_DEVICE)
         CloseDevice(joy_io_msg);

      if (close_flags & CL_JOY_STDIO)
         DeleteStdIO(joy_io_msg);

      if (close_flags & CL_REP_PORT)
         DeletePort (reply_port);

      if (close_flags & CL_JOY_PORT)
         DeletePort (joy_io_port);

      if (close_flags & CL_OUT_MEM)
         FreeMem (out_msg, sizeof (struct TlogMsg));

      if (close_flags & CL_JOY_BUF)
         FreeMem (joy_io_buff,
                  sizeof (struct InputEvent));
  }
      int
  joy_io_type (gameport_type)
          /* Set type of device attached to   */
          /* right game port.  Used in two    */
```

```
     /* parts of joymsg.                  */
  SHORT gameport_type;
{
  BYTE iobuff;

  joy_io_msg->io_Command = GPD_SETCTYPE;
  joy_io_msg->io_Length = 1;
  joy_io_msg->io_Data = (APTR) &iobuff;
  iobuff = gameport_type;

 /* Send the set controller type message to the */
 /* gameport device handler.                    */
  SendIO (joy_io_msg);
  WaitPort(joy_io_port);
 /* Check for errors, return any to caller */
  GetMsg(joy_io_port);
  return ((int) joy_io_msg->io_Error);
}
```

Listing 3. keymsg.c

```
/*-----------------------------------------+
|                keymsg.c                  |
| Program to read messages from console    |
| and send them to tlogger to be logged.   |
| Quits after gets and sends string defined|
| by TLOG_QUIT_MSG to tlogger.             |
| D.T.McClellan for AmigaWorld. Jan. 87    |
| See tlogger.h for running instructions.  |
+----------------------------------------*/

#include <stdio.h>
#include "tlogger.h"
  /* This defines the data structures used by */
  /*   the three programs, and incidentally   */
  /*   includes the needed exec .h files.     */

/*--------+
| Defines |
+--------*/

#define KEY_INIT_MSG     "KEYMSG starting up\n"
        /* Initial msg */
#define KEY_INIT_LEN     20
        /* Length of message include \0 */

struct MsgPort *reply_port;
        /* Port for tlogger to reply to */
struct MsgPort *out_port;
        /* tlogger's port */
struct TlogMsg *out_msg;
        /* Message to send */

main ()
{

/*----------------------------------+
| Get address of tlogger's port, send |
| startup message to it.             |
+----------------------------------*/

if ((out_port = FindPort (TLOG_PORT_NAME)) == NULL)
 {
  printf ("keymsg: Cannot find tlogger's port.\n");
  exit (-1);
 }


if ((reply_port = CreatePort (NULL, 0)) == NULL)
 {
  printf ("keymsg: Cannot allocate reply port\n");
  exit (-2);
 }

        /* Allocate and init message struct */
```

Hors d'oeuvres

Unique applications, tips and stuff

You may be using your Amiga at work, you may be using it at home, or you may be using it in the back seat of your car, but in some way or other, you are going to be using your Amiga in a slightly different way than anyone else. You are going to be running across little things that will help you to do something faster or easier or more elegantly.

AmigaWorld would like to share those shortcuts, ideas, unique applications, programming tips, things to avoid, things to try, etc., with everyone, and we'll reward you for your efforts with a colorful, appetizing, official *AmigaWorld* T-shirt. (Just remember to tell us your size.)

Send it in, no matter how outrageous, clever, obvious, humorous, subtle, stupid, awesome or bizarre. We will read anything, but we won't return it, so keep a copy for yourself. In cases of duplication, T-shirts are awarded on a first come, first serve basis.

So, put on your thinking berets and rush those suggestions to:

Hors d'oeuvres
***AmigaWorld* editorial**
80 Elm Street
Peterborough, NH 03458

[redacted]


```
if ((out_msg = (struct TlogMsg *) AllocMem (sizeof
      (struct TlogMsg), MEMF_CLEAR | MEMF_PUBLIC))
      == NULL)
 {
  printf ("keymsg: Insufficient memory
           for my message.\n");
  DeletePort (reply_port);
  exit (-3);
 }

        /* Initial message */
  out_msg->msghead.mn_Node.ln_Type = NT_MESSAGE;
  out_msg->msghead.mn_ReplyPort = reply_port;

  strcpy (out_msg->text, KEY_INIT_MSG);
  out_msg->msghead.mn_Length = KEY_INIT_LEN;
  PutMsg (out_port, out_msg);
  (void) WaitPort (reply_port);

/*-----------------------------------------------+
| Loop, reading messages from keyboard, sending  |
| them to tlogger.                               |
+-----------------------------------------------*/

for (;;)    /* Loop until user types in quit message */
 {
 printf ("Next Message > ");
 gets (out_msg->text);
 out_msg->msghead.mn_Length = strlen (out_msg->text);
 PutMsg (out_port, out_msg);
 (void) WaitPort (reply_port);

  if (! strncmp(out_msg->text, TLOG_QUIT_MSG,
       TLOG_QUIT_LEN))
      break;    /* Done! */

  } /* end for (;;) */

/*--------------+
| Close up shop|
+--------------*/

 DeletePort (reply_port);
 exit (0);
}
```

Listing 4. *tlogger.h*

```
/*---------------------------------------+
|               TLOGGER.H                |
| Specification of data structures       |
| and names used in the three programs   |
| of the Exec tutorial.                  |
| D.T.McClellan for AmigaWorld. Jan. 87  |
| Running Instructions at End.           |
+---------------------------------------*/

#include "exec/types.h"
#include "exec/exec.h"

/*--------+
| Defines |
+--------*/

#define TLOG_PORT_NAME"TLOG.PORT"
      /* Name of tlogger's port, for other */
      /* programs to send messages to.     */
#define TLOG_QUIT_MSG"quit"
      /* Message to enter to keymsg to     */
      /*   cause it and tlogger to stop. */
#define TLOG_QUIT_LEN4
      /* Length of TLOG_QUIT_MSG. */
#define TLOG_FILE_NAME"tlog.file"
      /* File log messages will be written to */
#define MAX_MSG_TEXT80
```

```
      /* Maximum length of text part of message. */

/*----------------+
| Data Structures |
+----------------*/

struct TlogMsg {
      /* Message format used by tlogger & callers */
      struct Message msghead; /* Standard header. */
      char    text[MAX_MSG_TEXT];
              /* Buffer space for text of message.*/
              /* Actual length of text is in     */
              /* msghead.mn_Length; usually will  */
              /* be less than MAX_MSG_TEXT.       */
};

/*----------------------+
| Commonly used Routines |
+----------------------*/

struct Task *FindTask();
struct MsgPort *CreatePort();
struct MsgPort *FindPort();
struct TlogMsg *WaitPort();
        /* Fib a little.  It returns a ptr       */
        /* to a generalized message; it          */
        /* only cares about the first few bytes. */
struct TlogMsg *GetMsg();
        /* Same "type" fib as for WaitPort.      */

APTR AllocMem();

/*-----------------------------------------------------
Running Instructions for tlogger, joymsg, and keymsg:

 1) These programs are meant to be run from the CLI;
    not Intuition.
 2) Have a joystick with a fire button attached
    to the right (rear) joystick port.
 3) CD to a directory on a disk it is ok to write on;
    for now let's assume that is DF1: and tlogger,
    keymsg, and joymsg are on DF1: as well.
 4) Issue the following commands to the CLI (commands
    typed in UPPERCASE, my notes about other things
    to do will be in lowercase).  These will cause
    tlogger and joymsg to become "background" tasks,
    and keymsg to be the foreground task interacting
    with you via the console.

      CD DF1:
      RUN TLOGGER
      RUN JOYMSG
 (hit the joystick button once or twice before keymsg)
      KEYMSG

 5) Now type in text at the keymsg "Next Message >"
    prompts, hitting a carriage return at the end of
    each message.  After 45 seconds or so of not
    hitting the joystick button, depress and release
    it four or five times.  After five total presses
    of the button, joymsg will exit (invisibly to
    you). Type some more text in at keymsg prompts.
    When ready to quit, type the word "quit" at one
    prompt (as the first and only four letters of
    that line). keymsg and tlogger will go away.

 6) TYPE TLOG.FILE
    This file contains the text of messages send to
    tlogger.  Look at the interspersing of ones send
    from keymsg and from joymsg; look for the button-up,
    button-down, and timeout messages from joymsg (it
    will send a timeout message to tlogger after 30
    seconds of inactive joystick buttons).

 7) Modify the programs as you want; be careful!
*/
```

from p. 14

CLI type:

COPY DF0: TO DF1: ALL

This reorganizes the file layout and improves the directory access times noticeably.

Noah Sherman
Olympia, WA

Changing Disks with One Drive

When using AmigaDOS and only one drive, things don't always work the way you might want them to. Especially when you have to replace the Workbench disk in the drive to load the command. Some AmigaDOS commands execute immediately once they are loaded, not giving you the chance to put the non-Workbench disk you wanted to work on back in the drive. Using the CD command doesn't work because, once loaded from the Workbench disk, it is executed instantly.

I found a way around this that people may not have discovered. When issuing one of those instant execution commands, I just add a space and question mark before pressing return, which results in the AmigaDOS version of on-line help. This will give you the syntax and format of any AmigaDOS command (a nice thing to remember for beginners) and also loads the command into memory before executing. This technique works with any of the AmigaDOS commands even if they don't have parameters, like the INFO command. Just type INFO ? and after the disk swap the message none: will appear. Replace the disk you want info on and press return.

I hope this makes things a bit easier for all of those other single-drivers like me.

Shawn Switenky
Prince Albert, Saskatchewan, Canada

Startup-Sequences

Startup Speedup

I would like to share a nifty little trick that will speed the startup-sequence on your Workbench disks. Most startup-sequences contain a number of ECHO statements that display bits of information and instructions. Each time the ECHO statement is encountered, it must be loaded from disk. This slows down execution time significantly. The statement ADDBUFFERS DF0: 15 as the first statement in your startup-sequence will eliminate this slowdown. The statement creates a sort of mini-RAM disk that stores the last disk access (or as much of it as will fit) for instant recall. You don't want to allocate too much RAM because once allocated, there is no way (that I know of) to get the RAM back. You can use this trick to allocate RAM storage for any drive by changing drive numbers in the ADDBUFFERS command. However, this command is only available under AmigaDOS 1.2.

I have also found that a foam place mat from the dinner table makes an excellent and inexpensive mouse pad.

R.D. Thomas
Millington, TN

Date/Time Startup

I found a way to enter the date and time without having to first go into the Preferences section. I edited the startup-sequence using ED. Change the startup-sequence to include the following lines

```
ECHO " "
ECHO "DEFAULT DATE:"        ; show last
```

```
  startup date
DATE
ECHO " "
ECHO "SET THE DATE AND TIME."
ECHO " DD-MMM-YY HH:MM"
ECHO " "
FAILAT 21       ; error trap
DATE ?
IF FAIL
ECHO " "
ECHO "PLEASE TRY AGAIN."
ECHO "USE SPACE TO SEPARATE
  DATE AND TIME."
ECHO " "
DATE ?
ENDIF
DATE
DATE TO SYS:S/LAST-STARTUP-DATE
  ; write a file
ECHO " "
```

Having the startup script request the date is an idea that has been around for a while, but this little sequence is unique in that it writes. As AmigaDOS boots up, it gets the date of the newest files on the Workbench disk. Writing a file when the clock is set during startup ensures that the next time the date is set, it will be set no earlier than the last time you used the computer. The file is written to the "s" directory. That it is named "Last-Startup-Date", even that it actually contains the date, makes no difference; these are merely coincidences of convenience.

One benefit of this startup is that you never have to type out the complete date unless you shamelessly ignore your Amiga for more than a week. The Date function recognizes day names as well as dates, and assumes that if you enter "Monday", you mean the next Monday. Thus, if you last booted on Friday, January 31, 1986, and you respond "23:12 Monday" to the date prompt, it is just as good as typing "23:12:00 03-mar-86", and somewhat easier. If you reboot during the day, just type the current time. As an aside, Date also recognizes "today", "yesterday", and "tomorrow".

Marcus Brooks
Austin, TX

Date/Time Sans Prompts

Many AmigaDOS commands generate their own prompts when they expect input. For example, the DATE ? command causes the prompt TIME,DATE,TO = VER/K to appear on the screen. You can suppress the command prompt by using the redirection operator > and the NIL device. Thus DATE >NIL: ? will send the prompt into computer never-never land without generating any error conditions. Just be sure to place the redirection operator before the parameter requester ?.

Roger Le Vine
Highland Park, IL

Startup RAM DOS

If you use the CLI a great deal and have only one disk drive, you can edit the startup-sequence to automatically create a RAM DOS for you and then leave you in the CLI without calling Workbench.

First, *make a backup copy of your Workbench disk!!!* This will enable you to start over in the event of a mishap. From the CLI prompt, type ED S/STARTUP-SEQUENCE. This will call up the editor program where you may then change the startup sequence. You can use the arrow keys to move around in the text; the CTRL and B keys let you delete lines. The ECHO command simply prints text on the screen so you can customize your startup screen to read anything you want. (For more information about the ED, see "Using the Amiga Editor" in the May/June 1986 issue of *AmigaWorld*.) Change ►

the startup-sequence to include the following lines:

```
MAKEDIR RAM:C
COPY C RAM:C
ASSIGN C: RAM:C
CD RAM:
```

Delete the last two lines of the startup-sequence (LoadWB and endcli >nil) if you do not want to enter Workbench after creating the RAM DOS. When everything looks the way you want it to, press the ESC key, then the X key, and then press Return. This will save your changes to the disk, replacing the original startup-sequence with your new version. (This is why you only want to try this on a backup copy of Workbench.) If you hopelessly mess things up, press the ESC key then the Q key to exit the ED without saving changes.

Now, when you warm or cold start your Amiga, the RAM DOS will automatically be created for you. For switching back and forth between disk "drives" (DF0: and the new RAM:), use the CD (change directory) command. For example, CD DF0: to get back to DF0: or CD RAM: to make the RAM disk the default directory.

Steve Worcester
Buena Park, CA

Startup INFO

Here is a way to use the INFO command in your startup-sequence to display, at startup, information about storage space on your mounted disks. Add these lines to your startup-sequence:

```
INFO
WAIT 5
```

The WAIT 5 line simply has the Amiga pause five seconds after displaying the information before going on with the rest of the startup-sequence; you can adjust the value to change the delay. This is quicker than having to open a CLI window after loading Workbench.

Steve Ehrhart
Wahiawa, HI

Auto CLI Window

I am a fifteen-year-old student living on Long Island and I have a tip that might be useful for people who like to use both the CLI and Workbench. First, *make a copy* of your Workbench as a precaution. From the CLI prompt type:

```
ED S/STARTUP–SEQUENCE
```

then move the cursor down and add the following line just *before* the last line (which reads ENDCLI >NIL:):

```
NEWCLI CON:540/150/100/50/CLI
```

Press the escape key, then the X key and return to save your modified startup-sequence. (If you mess things up hopelessly, press Q instead of X to bail out of the editing without saving your changes.)

This command opens a new CLI as a console window, which means that you must give the X and Y coordinates, the width and height of the window, and a title. The first two numbers in my line above (540 and 150) are the X and Y coordinates, which determine the placement of the window (in this case the lower right of the screen). The next two numbers (100 and 50) are the width and height of the window; the last part (CLI) is the name that you want to appear on the title bar of the new window. The Workbench screen is in the 640 × 200-pixel mode, so consider this if you want to change the location of the window; note that, in my example, 540 + 100 = 640 (screen pixel width), and 150 + 50 = 200 (screen pixel height).

Once you have altered the startup-sequence and rebooted, you will be in Work-

bench, but a small CLI window will be automatically opened in the lower right of the screen. The advantage of this method is that you don't have to have quick fingers for a CTRL-D, or lots of patience for opening drawers and windows to get to a CLI. And when you want to go back to Workbench from the CLI, you don't have to type LOADWB return ENDCLI as you would with other variations of startup-sequences.

Michael Rubino
Commack, NY

Advanced CLI

Alternate Character Sets

Almost every person who has played around with the CLI has seen the alternate character set. You hit the wrong key or try something new and. . . BOOM! These funny looking characters appear instead of the normal character set. While fooling around the other day, I discovered two very useful keystrokes: CTRL O (the letter "O," not the number zero) will turn on the normal character set and CTRL N will turn on the alternate character set. So, if you do something funny and see the alternate character set filling your screen, hit CTRL O and you will be back in business.

John D. Hamilton
Fresno, CA

Custom CLI Modes

From the CLI you can enable the various text modes with a few short keystrokes. The sequence is simply:

ESC[*n*1;*n*2;*n*3m

where ESC is the escape key, *n*1 is a number for style, *n*2 is a number for foreground color, *n*3 is a number for background color, and m is the sequence terminator. The following is a list of values you can use, but be aware that if the colors have been changed through Preferences, these numbers will give different results.

STYLE
- 0 = Plain Text
- 1 = Bold Face
- 3 = Italic
- 4 = Underline
- 7 = Inverse

FOREGROUND
- 30 = Default
- 31 = White
- 32 = Binary complement color (black for the Workbench default colors)
- 33 = Red

BACKGROUND
- 40 = Default
- 41 = White
- 42 = Binary complement color
- 43 = Red

You can also combine styles by entering each style number separated by a semicolon. For example

ESC[1;33;41m

will give boldface red on white, and

ESC[1;3;4;31m

will give boldface underlined italics in white.

Some of you will notice that these are the same escape sequences used by the printer.device. [*The Technical Reference Guide in this issue has a complete list of printer sequences.*] Since the CON: is treated pretty much the same way as the PRT:, some of the printer sequences work on the screen as well as on paper.

Aside from being just fun, there are some practical uses for these customizing tech- ►

niques. You can include these escape sequences as part of your file comments. I use them in commenting my directory names (which I like to keep as short as possible to avoide extra typing). When you list your files, you can make the ones you chose stand out from the others. For example:

FILENOTE *filename* COMMENT
"this file is ESC[1m IMPORTANT! ESC[0m"

prints the word IMPORTANT in boldface when listed. Filenote is the DOS command to append comments to a file. Since we're entering escape sequences and spaces in our comment, we need the quotes. It is also important to change the text back to normal at the end with ESC[0m or everything listed after that will have the same style and color.

Jim Davis
Niles, IL

Custom Prompts and Echoes

Both the PROMPT and ECHO commands found in the C directory can be modified to issue some of the ANSI command characters in order to change the displayed text both in color and style. By using the same techniques mentioned above, you can add color to the 1> CLI prompt or anything "echoed" to the screen.

To tell PROMPT or ECHO to accept the codes, a template of commands is used. This template is:

CSIstyle;foreground; backgroundm

CSI is Control Sequence Introducer, which can be ESC [or in hex 1B 5B; for the PROMPT and ECHO commands, it is *e. The style and color options are listed above. These colors will be different if changed through Preferences. The m at the end is the Command sequence terminator. Once the PROMPT and ECHO commands have been modified, they stay that way until you either change them again or re-boot the system. To modify the prompt, try something like: PROMPT "*e[3;33;41mAmigaWorld> *e[0m". You should get an italic *AmigaWorld>* in red on a white background prompt with everything you type after it in normal white on default blue. To modify the ECHO command, try something like: ECHO "*e[4;32mHello There *e[0m". You should get the underlined words "Hello There" printed in black.

David Morgan
Robina, Australia

Pattern Matching

AmigaDOS does not have an explicit wildcard string like the asterisk (*) in MS-DOS. However, a wildcard string can be implemented by using the pattern-matching pair #?. In AmigaDOS the ? is the wildcard symbol for a single character, while the # means match the following pattern multiple times. For example, the command:

COPY DF0:SAM/TEST.#? TO DF1:SAM[

would copy all the files that begin with TEST. in the SAM directory on the disk in drive 0 to the SAM directory on the disk in drive 1. However, there is a subtle difference between #? and *. Under MS-DOS, the period is a delimiter between the file name and the suffix, but it is not used the same way under AmigaDOS. So, in the previous example, all files named TEST with a suffix would be copied, but the file TEST *itself* would not be copied, since it was not followed by a period.

Sam Spear
Fort Worth, TX

Programming

SAY from BASIC

If you want to use the Say (voice) function, the best method is to store the spoken text in a sequential data file created with the built-in Ed function or any word processor that allows an ASCII save. After the file is created, add the following lines to your Amiga Basic program:

```
OPEN file name FOR INPUT AS #1
  REM file name CREATED USING ED
WHILE NOT EOF(1)
LINE INPUT#1,A$
SAY TRANSLATE$(A$)
WEND
CLOSE #1
```

You can listen to the spoken text before placing it in your program by opening a CLI window and typing SAY -X *file name.*

Karl Dittman
Salem, MO

Big BASIC Programs Loader

This tidbit has been around for awhile, but I havn't seen it in any of the magazines yet. I'm sure that there are a lot of readers like me who do most of their programming in BASIC. Many were expecting Workbench 1.2 to allow more BASIC program space than the 25K in 1.1, but that apparently

didn't happen. I, for one, couldn't wait. I have a program that uses almost 100K, which doesn't fit in Amiga Basic unless you "trick" it. If you have a program that will exceed 25K, first write a "loader" program that will increase the available space, and then "chain" your main program. For example, the loader program that I use is:

```
Clear,3000
Clear,100000
```

where 100000 bytes are made available

CHAIN *main program name*

This opens the required memory and then "carries" it into the main program.

Russell Strong
Richmond, TX

Amiga Basic Joysticks

While trying to use the joystick functions STICK and STRIG in Amiga Basic, I found some errors in the documentation (pp. 8-143 and 8-144). The manual states that return values for the Y directions on both A and B joysticks are 1 for the "up" direction and −1 for the "down" direction. Actually it is the other way around. A return value of 1 means down and a return value of −1 means up. The X direction functions are described correctly in the manual.

The STRIG function explanation has two mistakes. First, the return value stated for "depressed fire button" is 1 when it should be −1. Second, the subscripts to acquire status from B joystick should read STRIG (2) and STRIG (3), not STRIG (0) and STRIG (1).

This was aggravating to figure out, so I thought I would share this information with others who may want to use these functions in their own programs.

Rob Fallier
Vidalia, GA

Clearing Keyboard Buffer in BASIC

The ability to type ahead on the Amiga is usually a good thing to have, but it can sure mess up the INKEY$ statement in a BASIC program. The following subroutine will make sure no old, garbage keypresses are mistaken for a response.

```
SUB CLEARKEYS STATIC
FOR X = 1 TO 10
r$ = INKEY$
NEXT X
ENDSUB
```

To use, call it immediately before the INKEY$ statement. Example:

```
CALL CLEARKEYS
WHILE INKEY$ = "": WEND
```

Margaret Hettinger
Lebanon Junction, KY

Point Finding in Amiga Basic

Here is a tidbit I wrote that is very useful to me as an Amiga Basic programmer. At times I need to know the location of specific points on the screen. I allow my main program to draw objects on the screen, then STOP the program and run this little routine. Clicking the left-mouse button produces the coordinates. This way I can easily find the vertices of any object on the screen.

```
POINTFIND:
IF MOUSE(0) = 1 THEN
   PRINT MOUSE(1),MOUSE(2)
END IF
GOTO POINTFIND
```

MOUSE(1) returns the X coordinate and MOUSE(2) returns the Y coordinate.

Jeff Kerschner
Berwick, ME

C Compiler Printouts

I was having difficulty compiling a program in C when I thought how nice it would be to have a hard copy of the compiler errors to discuss with other programmers. The command format for Manx's Aztec C compiler is:

```
cc <-option> filename
```

Thinking this chore would be easy, I tried to redirect the console output to the printer by typing:

```
cc filename > prt:
```

However, the compiler treated > prt: as a parameter, since options can appear either before or after the name of the C source file that resulted in an error. Due to a quirk of the system, prt: should precede compiler invocation so that:

```
cc > prt: <-option> filename
```

will properly compile *filename* and redirect output to the printer and return output to the console after compilation is executed.

Sam Spear
Fort Worth, TX

Long Variables in Aztec C

Having trouble getting your Aztec C programs to run correctly? Are you using library functions? The problem could be with ►

variables of the type LONG. Aztec C differs from Lattice C in that it's integers are not automatically of type LONG. So whenever you fill a parameter to a library function call with a number (not a variable), the compiler assumes you want a SHORT. However the library function may be expecting a LONG. When this occurs, you can easily crash the machine or get unexpected results. The solution is to put a capital L after any number you specify in a function call (which forces the number to be of type LONG), or to make a variable for that parameter and declare it to be of type LONG.

For example, I had a hard time getting the ScrollRaster function to work until I realized that I had not defined the parameters of the function as type LONG:

ScrollRaster(RPort,1,0,10,10,100,100);
won't work, but

ScrollRaster(RPort,1L,0L,10L,10L,
100L,100L); will work.

Check the *Intuition Reference Manual* or the *ROM Kernal Manual* whenever you use a function to see if your parameters are of the correct type.This is a major reason why those Lattice C source files just don't seem to work with Aztec C.

Nicholas Barrowman
Kingston, Ontario, Canada

Hors d'oeuvres

Unique applications, tips and stuff

You may be using your Amiga at work, at home, or in the back seat of your car, but somehow you'll be using it in a unique way. You will discover things that will let you do something faster, easier or more elegantly.

AmigaWorld would like to share those shortcuts, ideas, things to avoid, things to try, etc., with everyone, and we'll reward you with a colorful, appetizing, official *AmigaWorld* T-shirt. (Just remember to tell us your size.)

Send it in, no matter how outrageous, clever, humorous or bizarre. We will read anything, but we won't return it, so keep a copy for yourself. In cases of duplication, T-shirts are awarded on a first come, first serve basis.

So, put on your thinking berets and rush those suggestions to:

Hors d'oeuvres
***AmigaWorld* editorial**
80 Elm Street
Peterborough, NH 03458

Reading Joysticks in C

For my latest C language project, I needed a way to read the joystick ports. The manuals propose a very complicated procedure, so I developed my own. In my routine, I step down into the machine and read the hardware registers directly. Here is a short routine that should run on both Aztec and Lattice compilers:

```
#define PORT 2
short *joy = 0xdff008 + 2 * port;
char *cia = 0xbfe001;
main() {
  for(;;) {
    if (*joy & 2)
      printf("RIGHT . . .");
    if (*joy & 512)
      printf("LEFT . . .");
    if ((*joy >> 1 ^ *joy) & 1)
      printf("DOWN . . .");
    if ((*joy >> 1 ^ *joy) & 256)
      printf("UP . . .");
    if (!(*cia & 64 * PORT))
      printf("FIRE !!!");
    printf("\n");
    }
}
```

Martin Brenner
Neu-Ulm, West Germany ■

The Pull-Down Menu

NO SCAN LINES! TOP QUALITY—*FAST*

SLIDES

ANY IFF FILE PROCESSED DIRECTLY FROM YOUR DISK!
2k resolution 35mm slides $9.30 each. Also digital color separations.
Now accepting Visa/MC, minimum order $25.00.
Call or write for our full service list:

ImageSet—555 19th St., San Francisco, CA 94107
(415) 626-8366

THE DISK SPECIALISTS SPECIAL OFFER

MCP

3.5″ DSDD, Fujitsu, Bulk, Lifetime Guarantee
• 50 Disks (1 carton), @ $1.15 per disk •
Limit 2 cartons per customer. Write/Call for larger quantities, other product listings.

MCP Associates, P.O. Box 6260, LIC, NY 11106
(718) 956-9000

"DISNEY" STYLE ANIMATION $295

- Create 3D characters and scenes quickly and easily.
- Includes shading in 4096 colors, background matting.
- Real-time audition.
- Animation vignettes complete overnight.

Hash Enterprises
14201 SE 16th Circle, Vancouver WA 98684
(206) 256-8567

HATE DUSTCOVERS? LOVE COVERUP OR MONEY BACK!

COVERUP attaches with velcro. It flips out of the way while you work, but stays handy. **COVERUP** is made of non-static fabric. A set (monitor, CPU, ext. drive) costs $25, order direct from:

COVERUP (805) 487-1545
3129 Lassen Street
Oxnard, CA 93033

Dealer Inquiry Welcome

MicroEd

Educational Software K thru ADULT

SEND FOR A LIST OF OUR 29 AMIGA PROGRAMS. ALL USE VOICE. MANY USE DIGITIZED PICTURES.

MicroEd, Incorporated
P.O. Box 24750
Edina, MN 55424

Tek 4105 and VT100 Emulation

MiddleMan™ $59.95 — True VT100 emulation. Fully integrated into the Amiga multi-tasking environment. Text file transfer, macros, 7000 baud text DISPLAY rate, graphics character set, and more. Available since July 1986.

MiddleMan II™ $149.95 — All the features of MiddleMan™ plus Tektronics® 4105 color graphics terminal emulation. Available early Fall, 1987. Pre-release copies available NOW for $129.95.

1-800-6 BCP, Inc. In Alabama: (205) 881-1110 VISA MasterCard
Benaiah Computer Products, Inc. P.O. Box 14502 Huntsville, AL 35815

The Pull-Down Menu

AMIGAWORLD'S New Pull-Down menu is a great opportunity for those with AMIGA products to reach over 70,000 Amiga owners. AmigaWorld is the only publication with a subscription card in the box with every Amiga computer, national newsstand distribution by ICD Hearst, and single copy sales in computer stores carrying the AMIGA as well as large bookstores such as B. Dalton and Walden Books.

To reserve your Pull-Down Menu ad call Heather Paquette on the East Coast at 1-800-441-4403 or Brenda Billhimer on the West Coast at 1-415-328-3471. We accept checks, money orders, MasterCard or VISA.

MEGAmiga

0 wait, 2 megabyte ram with own power supply and pass through bus for $595 ppd.

CALL (602) 622-1344
API, 1620 N. Park Ave, Tuscon AZ 85719
Dealer Inquires Welcome

AMIGA T-SHIRTS AND SWEATSHIRTS

Quality white shirts silk screened with Amiga logo in beautiful color.

T-shirts: $12.00 **Sweatshirts: $18.50**
In CA, add 6% tax.

SEND CHECK OR MONEY ORDER TO:
T's Me, P.O. Box 11746, Santa Ana, CA 92711
2525 Shadow Lake, Santa Ana, CA 92701
Call (714) 639-6545
Dealer Inquiries Invited
Amiga is a trademark of Commodore-Amiga, Inc.

Please allow 3 to 6 weeks for delivery

USE YOUR PHOTOS

Slides or Drawings with your Amiga
PROGRAMS, BUSINESS REPORTS, SCHOOL PAPERS and COMPUTER ART

Images digitized and returned on disk, using standard IFF format. Images cropped unless full frame specified. Satisfaction guaranteed. Send copies; we cannot be responsible for loss or damage of materials. One image—$15; up to four images—$18; up to seven images—$21; each additional image—$2. Add $2 shipping/handling. (CA res. add 6% sales tax. Foreign add $4 US) **Black and White digitizing is available for the Commodore 64/128**
Send check or money order to:

YORE PHOTO™, Dept. D, PO Box 1269, 29 Palms, CA 92277
(619) 367-3779

In a Hurry? Specify **'ZAP!SHOT'**. Up to seven images will be digitized and in return Express Mail within 24 hours of our receipt (Monday–Friday only). Add $20 special postage and handling.

AmigaWorld is a member of the CW Communications/Inc. group, the world's largest publisher of computer-related information. The group publishes over 80 computer publications in more than 28 major countries. Fourteen million people read one or more of the group's publications each month. Members of the CWCI group contribute to the *CW International News Service* offering the latest on domestic and international computer news. Members of the group include: ARGENTINA'S *Computerworld/Argentina, PC Mundo*; ASIA'S *Computerworld Hong Kong, Computerworld SE Asia*; AUSTRALIA'S *Computerworld Australia, Communications World, Australian PC World, Australian Macworld*; AUSTRIA'S *Computerwelt Oesterreich*; BRAZIL'S *DataNews, PC Mundo, Micro Mundo*; CHILE'S *Informatica, Computacion Personal*; DENMARK'S *Computerworld Danmark, PC World Danmark*; FINLAND'S *Tietoviikko, Mikro*; FRANCE'S *Le Monde Informatique, Distributique, InfoPC, Le Monde Des Telecoms*; GREECE'S *Micro and Computer Age*; HUNGARY'S *Computerworld/SZT, Mikrovilag, Tudomany*; INDIA'S *Dataquest*; ISRAEL'S *People & Computers Weekly, People & Computers Monthly*; ITALY'S *Computerworld Italia*; JAPAN'S *Computerworld Japan*; MEXICO'S *Computerworld Mexico, Compumundo*; THE NETHERLANDS' *Computerworld Netherlands, PC World Netherlands*; NEW ZEALAND'S *Computerworld New Zealand*; NORWAY'S *Computerworld Norge, PC World Norge*; PEOPLE'S REPUBLIC OF CHINA'S *China Computerworld, China Computerworld Monthly*; SAUDI ARABIA'S *Arabian Computer News*; SOUTH KOREA'S *Computerworld Korea, PC World Korea*; SPAIN'S *Computerworld Espana, Commodore World, PC World Espana, Comunicaciones World, Informatica Industria*; SWEDEN'S *ComputerSweden, Mikrodatorn, PC World*; SWITZERLAND'S *Computerworld Schweiz*; UNITED KINGDOM'S *Computer News, DEC Today, ICL Today, PC Business World, Lotus UK*; UNITED STATES' *AmigaWorld, Boston Computer News, CD-ROM Review, Computerworld, Computers in Science, Digital News, Federal Computer Week, 80 Micro, FOCUS Publications, inCider, InfoWorld, MacWorld, Micro Marketworld, Network World, PC World, Portable Computer Review, Publish!, PC Resource, RUN*; VENEZUELA'S *Computerworld Venezuela*; WEST GERMANY'S *Computerwoche, PC Welt, Run, Information Management, PC Woche.*

Manuscripts: Contributions in the form of manuscripts with drawings and/or photographs are welcome and will be considered for possible publication. *AmigaWorld* assumes no responsibility for loss or damage to any material. Please enclose a self-addressed, stamped envelope with each submission. Payment for the use of any unsolicited material will be made upon acceptance. All contributions and editorial correspondence (typed and double-spaced, please) should be directed to *AmigaWorld* Editorial Offices, 80 Elm Street, Peterborough, NH 03458; telephone: 603-924-9471. **Advertising Inquiries** should be directed to Advertising Offices, CW Communications/Peterborough, Inc., 80 Elm Street, Peterborough, NH 03458; telephone: 800-441-4403. **Subscription problems or address changes:** Call 1-800-227-5782 or write to *AmigaWorld*, Subscription Department, PO Box 868, Farmingdale, NY 11737. **Problems with advertisers:** Send a description of the problem and your current address to: *AmigaWorld*, 80 Elm Street, Peterborough, NH 03458, ATTN.: Cynde Garrett Customer Service Manager, or call 1-800-441-4403.

List of Advertisers

* This advertiser prefers to be contacted directly

This index is provided as an additional service. The publisher does not assume liability for errors or omissions.

TO RECEIVE MORE INFORMATION

AMIGA WORLD

TEAR
out the perforated card. Please print or type your name and address where indicated.

CIRCLE
the numbers on the card that correspond to the reader service numbers on the advertisements that interest you.

ORDER
a one year subscription to AmigaWorld by circling 500 on the card.

MAIL
the card with your check, money order or U.S. currency to:
AmigaWorld
Reader Service Dept.
P.O. Box 363
Dalton, MA 01227
Or, you may request billing.

LOOK FOR
your subscription in 10 to 12 weeks.

REMEMBER
to put the proper postage on the card.

READER SERVICE

SPECIAL ISSUE 1987

This card valid until December 31, 1987

☐ Mr.
☐ Mrs.
☐ Ms. Name ______________________

Title ______________________

Address ______________________

City/State/Zip ______________________

Telephone ______________________

A. How would you rate this issue of AmigaWorld? (pick one)
☐ 1. GREAT! ☐ 2. Very Good ☐ 3. Pretty Good ☐ 4. Good ☐ 5. Fair ☐ 6. Poor ☐ 7. Very Poor ☐ 8. Terrible

B. What will be your next major peripheral purchase?
☐ 1. Monitor ☐ 2. Printer ☐ 3. Modem ☐ 4. Memory Expansion ☐ 5. Disk Drive (hard or floppy) ☐ 6. Sidecar ☐ 7. Gen Lock or Frame Grabber ☐ 8. Music (Midi, Keyboard, etc.) ☐ 9. Other

C. Check all of the endings that best complete this sentence: "Most of AmigaWorld is . . ."
☐ 1. Just Right ☐ 2. Too Simple ☐ 3. Too Complex ☐ 4. Fluff ☐ 5. Useful ☐ 6. Useless ☐ 7. Interesting ☐ 8. Biased ☐ 9. Invaluable

D. What topics would you like to see covered in future issues of AmigaWorld? (Please pick three.)
☐ 1. C Language ☐ 2. Amiga Basic ☐ 3. CLI ☐ 4. Telecommunications ☐ 5. Business Applications ☐ 6. IBM Compatibility ☐ 7. Home Applications ☐ 8. Education ☐ 9. Video ☐ 10. Science and Engineering ☐ 11. How others use the Amiga ☐ 12. Buyer's Guides ☐ 13. Comparative Reviews ☐ 14. Music ☐ 15. Graphics ☐ 16. Program Listings ☐ 17. New Products ☐ 18. Opinions ☐ 19. Hardware Projects ☐ 20. Other

E. What are your favorite things about AmigaWorld? (Please pick all that apply.)
☐ 1. Zeitgeist (Editor's Page) ☐ 2. Repartee (Letters) ☐ 3. Interviews ☐ 4. Help Key (Questions) ☐ 5. Features ☐ 6. Tutorials ☐ 7. Hors d'oeuvres (hints/tips) ☐ 8. Advertisements ☐ 9. Reviews ☐ 10. News ☐ 11. Digital Canvas ☐ 12. Articles ☐ 13. Covers ☐ 14. Reader Service Card ☐ 15. Everything

F. Which of the following categories do you plan to purchase software from in the next 12 months?
☐ 1. Entertainment ☐ 2. Word Processing ☐ 3. Communications ☐ 4. Spreadsheets ☐ 5. Home Productivity ☐ 6. Programming ☐ 7. Software Development ☐ 8. Video/Graphics Creation ☐ 9. Database Management ☐ 10. Financial Management ☐ 11. Graphics ☐ 12. Education ☐ 13. Music ☐ 14. Hardware Development ☐ 15. Sound/Speech Development ☐ 16. CAD/CAM

G. Have you ever purchased a product after receiving the information you've requested from an AmigaWorld reader service card?
☐ 1. Yes ☐ 2. No

H. Where do you buy your computer products? (Please pick one.)
☐ 1. Computer Dealer ☐ 2. Mail Order ☐ 3. Manufacturer ☐ 4. Discount/Department Store ☐ 5. Other ______

I. Do you own an Amiga?
☐ 1. Yes ☐ 2. No

J. Where do you use your Amiga?
☐ 1. Home ☐ 2. Work ☐ 3. School ☐ 4. At home for business ☐ 5. Both at home and work ☐ 6. Both at home and school ☐ 7. I don't use an Amiga

K. Is this your copy of AmigaWorld?
☐ 1. Yes ☐ 2. No

L. If you are not a subscriber, please circle 499.

M. If you would like a one year subecription to AmigaWorld (six issues), please circle 500 on this card. Each subscription is $14.97. (Canada & Mexico, $17.97, Foreign Surface $34.97, one year only—U.S. BANK FUNDS ONLY.) Please allow 10-12 weeks for delivery.

1	6	11	16	21	201	206	211	216	221	401	406	411	416	421
2	7	12	17	22	202	207	212	217	222	402	407	412	417	422
3	8	13	18	23	203	208	213	218	223	403	408	413	418	423
4	9	14	19	24	204	209	214	219	224	404	409	414	419	424
5	10	15	20	25	205	210	215	220	225	405	410	415	420	425
26	31	36	41	46	226	231	236	241	246	426	431	436	441	446
27	32	37	42	47	227	232	237	242	247	427	432	437	442	447
28	33	38	43	48	228	233	238	243	248	428	433	438	443	448
29	34	39	44	49	229	234	239	244	249	429	434	439	444	449
30	35	40	45	50	230	235	240	245	250	430	435	440	445	450
51	56	61	66	71	251	256	261	266	271	451	456	461	466	471
52	57	62	67	72	252	257	262	267	272	452	457	462	467	472
53	58	63	68	73	253	258	263	268	273	453	458	463	468	473
54	59	64	69	74	254	259	264	269	274	454	459	464	469	474
55	60	65	70	75	255	260	265	270	275	455	460	465	470	475
76	81	86	91	96	276	281	286	291	296	476	481	486	491	496
77	82	87	92	97	277	282	287	292	297	477	482	487	492	497
78	83	88	93	98	278	283	288	293	298	478	483	488	493	498
79	84	89	94	99	279	284	289	294	299	479	484	489	494	499
80	85	90	95	100	280	285	290	295	300	480	485	490	495	500
101	106	111	116	121	301	306	311	316	321	501	506	511	516	521
102	107	112	117	122	302	307	312	317	322	502	507	512	517	522
103	108	113	118	123	303	308	313	318	323	503	508	513	518	523
104	109	114	119	124	304	309	314	319	324	504	509	514	519	524
105	110	115	120	125	305	310	315	320	325	505	510	515	520	525
126	131	136	141	146	326	331	336	341	346	526	531	536	541	546
127	132	137	142	147	327	332	337	342	347	527	532	537	542	547
128	133	138	143	148	328	333	338	343	348	528	533	538	543	548
129	134	139	144	149	329	334	339	344	349	529	534	539	544	549
130	135	140	145	150	330	335	340	345	350	530	535	540	545	550
151	156	161	166	171	351	356	361	366	371	551	556	561	566	571
152	157	162	167	172	352	357	362	367	372	552	557	562	567	572
153	158	163	168	173	353	358	363	368	373	553	558	563	568	573
154	159	164	169	174	354	359	364	369	374	554	559	564	569	574
155	160	165	170	175	355	360	365	370	375	555	560	565	570	575
176	181	186	191	196	376	381	386	391	396	576	581	586	591	596
177	182	187	192	197	377	382	387	392	397	577	582	587	592	597
178	183	188	193	198	378	383	388	393	398	578	583	588	593	598
179	184	189	194	199	379	384	389	394	399	579	584	589	594	599
180	185	190	195	200	380	385	390	395	400	580	585	590	595	600

READER SERVICE

SPECIAL ISSUE 1987

This card valid until December 31, 1987

☐ Mr.
☐ Mrs.
☐ Ms. Name ______________________

Title ______________________

Address ______________________

City/State/Zip ______________________

Telephone ______________________

A. How would you rate this issue of AmigaWorld? (pick one)
☐ 1. GREAT! ☐ 2. Very Good ☐ 3. Pretty Good ☐ 4. Good ☐ 5. Fair ☐ 6. Poor ☐ 7. Very Poor ☐ 8. Terrible

B. What will be your next major peripheral purchase?
☐ 1. Monitor ☐ 2. Printer ☐ 3. Modem ☐ 4. Memory Expansion ☐ 5. Disk Drive (hard or floppy) ☐ 6. Sidecar ☐ 7. Gen Lock or Frame Grabber ☐ 8. Music (Midi, Keyboard, etc.) ☐ 9. Other

C. Check all of the endings that best complete this sentence: "Most of AmigaWorld is . . ."
☐ 1. Just Right ☐ 2. Too Simple ☐ 3. Too Complex ☐ 4. Fluff ☐ 5. Useful ☐ 6. Useless ☐ 7. Interesting ☐ 8. Biased ☐ 9. Invaluable

D. What topics would you like to see covered in future issues of AmigaWorld? (Please pick three.)
☐ 1. C Language ☐ 2. Amiga Basic ☐ 3. CLI ☐ 4. Telecommunications ☐ 5. Business Applications ☐ 6. IBM Compatibility ☐ 7. Home Applications ☐ 8. Education ☐ 9. Video ☐ 10. Science and Engineering ☐ 11. How others use the Amiga ☐ 12. Buyer's Guides ☐ 13. Comparative Reviews ☐ 14. Music ☐ 15. Graphics ☐ 16. Program Listings ☐ 17. New Products ☐ 18. Opinions ☐ 19. Hardware Projects ☐ 20. Other

E. What are your favorite things about AmigaWorld? (Please pick all that apply.)
☐ 1. Zeitgeist (Editor's Page) ☐ 2. Repartee (Letters) ☐ 3. Interviews ☐ 4. Help Key (Questions) ☐ 5. Features ☐ 6. Tutorials ☐ 7. Hors d'oeuvres (hints/tips) ☐ 8. Advertisements ☐ 9. Reviews ☐ 10. News ☐ 11. Digital Canvas ☐ 12. Articles ☐ 13. Covers ☐ 14. Reader Service Card ☐ 15. Everything

F. Which of the following categories do you plan to purchase software from in the next 12 months?
☐ 1. Entertainment ☐ 2. Word Processing ☐ 3. Communications ☐ 4. Spreadsheets ☐ 5. Home Productivity ☐ 6. Programming ☐ 7. Software Development ☐ 8. Video/Graphics Creation ☐ 9. Database Management ☐ 10. Financial Management ☐ 11. Graphics ☐ 12. Education ☐ 13. Music ☐ 14. Hardware Development ☐ 15. Sound/Speech Development ☐ 16. CAD/CAM

G. Have you ever purchased a product after receiving the information you've requested from an AmigaWorld reader service card?
☐ 1. Yes ☐ 2. No

H. Where do you buy your computer products? (Please pick one.)
☐ 1. Computer Dealer ☐ 2. Mail Order ☐ 3. Manufacturer ☐ 4. Discount/Department Store ☐ 5. Other ______

I. Do you own an Amiga?
☐ 1. Yes ☐ 2. No

J. Where do you use your Amiga?
☐ 1. Home ☐ 2. Work ☐ 3. School ☐ 4. At home for business ☐ 5. Both at home and work ☐ 6. Both at home and school ☐ 7. I don't use an Amiga

K. Is this your copy of AmigaWorld?
☐ 1. Yes ☐ 2. No

L. If you are not a subscriber, please circle 499.

M. If you would like a one year subscription to AmigaWorld (six issues), please circle 500 on this card. Each subscription is $14.97. (Canada & Mexico, $17.97, Foreign Surface $34.97, one year only—U.S. BANK FUNDS ONLY.) Please allow 10-12 weeks for delivery.

1	6	11	16	21	201	206	211	216	221	401	406	411	416	421
2	7	12	17	22	202	207	212	217	222	402	407	412	417	422
3	8	13	18	23	203	208	213	218	223	403	408	413	418	423
4	9	14	19	24	204	209	214	219	224	404	409	414	419	424
5	10	15	20	25	205	210	215	220	225	405	410	415	420	425
26	31	36	41	46	226	231	236	241	246	426	431	436	441	446
27	32	37	42	47	227	232	237	242	247	427	432	437	442	447
28	33	38	43	48	228	233	238	243	248	428	433	438	443	448
29	34	39	44	49	229	234	239	244	249	429	434	439	444	449
30	35	40	45	50	230	235	240	245	250	430	435	440	445	450
51	56	61	66	71	251	256	261	266	271	451	456	461	466	471
52	57	62	67	72	252	257	262	267	272	452	457	462	467	472
53	58	63	68	73	253	258	263	268	273	453	458	463	468	473
54	59	64	69	74	254	259	264	269	274	454	459	464	469	474
55	60	65	70	75	255	260	265	270	275	455	460	465	470	475
76	81	86	91	96	276	281	286	291	296	476	481	486	491	496
77	82	87	92	97	277	282	287	292	297	477	482	487	492	497
78	83	88	93	98	278	283	288	293	298	478	483	488	493	498
79	84	89	94	99	279	284	289	294	299	479	484	489	494	499
80	85	90	95	100	280	285	290	295	300	480	485	490	495	500
101	106	111	116	121	301	306	311	316	321	501	506	511	516	521
102	107	112	117	122	302	307	312	317	322	502	507	512	517	522
103	108	113	118	123	303	308	313	318	323	503	508	513	518	523
104	109	114	119	124	304	309	314	319	324	504	509	514	519	524
105	110	115	120	125	305	310	315	320	325	505	510	515	520	525
126	131	136	141	146	326	331	336	341	346	526	531	536	541	546
127	132	137	142	147	327	332	337	342	347	527	532	537	542	547
128	133	138	143	148	328	333	338	343	348	528	533	538	543	548
129	134	139	144	149	329	334	339	344	349	529	534	539	544	549
130	135	140	145	150	330	335	340	345	350	530	535	540	545	550
151	156	161	166	171	351	356	361	366	371	551	556	561	566	571
152	157	162	167	172	352	357	362	367	372	552	557	562	567	572
153	158	163	168	173	353	358	363	368	373	553	558	563	568	573
154	159	164	169	174	354	359	364	369	374	554	559	564	569	574
155	160	165	170	175	355	360	365	370	375	555	560	565	570	575
176	181	186	191	196	376	381	386	391	396	576	581	586	591	596
177	182	187	192	197	377	382	387	392	397	577	582	587	592	597
178	183	188	193	198	378	383	388	393	398	578	583	588	593	598
179	184	189	194	199	379	384	389	394	399	579	584	589	594	599
180	185	190	195	200	380	385	390	395	400	580	585	590	595	600

READER SERVICE

TO RECEIVE MORE INFORMATION

AMIGA WORLD

TEAR
out the perforated card. Please print or type your name and address where indicated.

CIRCLE
the numbers on the card that correspond to the reader service numbers on the advertisements that interest you.

ORDER
a one year subscription to AmigaWorld by circling 500 on the card.

MAIL
the card with your check, money order or U.S. currency to:
AmigaWorld
Reader Service Dept.
P.O. Box 363
Dalton, MA 01227
Or, you may request billing.

LOOK FOR
your subscription in 10 to 12 weeks.

REMEMBER
to put the proper postage on the card.

READER SERVICE

PLACE STAMP HERE

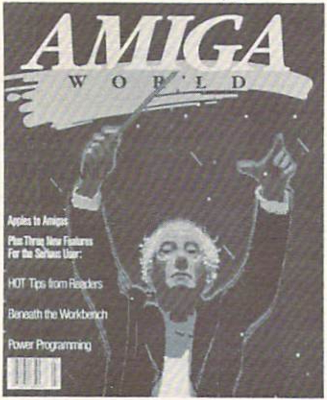
AmigaWorld
ATTN: Reader Service Dept.
P.O. Box 363
Dalton, MA 01227

PLACE STAMP HERE

AmigaWorld
ATTN: Reader Service Dept.
P.O. Box 363
Dalton, MA 01227

CHARTER OFFER

Save 25%



Enter my one year subscription (6 issues) to *AmigaWorld* for the low charter subscription price of $14.97. Money back guarantee: If I'm not satisfied at any time I will receive a full refund.

☐ Payment Enclosed
☐ Bill Me

Name ______________________

Address ______________________

City ______________ State ______ Zip ______

Please make check payable to *AmigaWorld*. Canada & Mexico $17.97, 1 year only, US Funds drawn on US Bank. Foreign Surface $34.97, 1 year only, US Funds drawn on US Bank. Foreign Airmail please inquire. Please allow 6–8 weeks for delivery. This offer is for new subscribers only!

37SB2C

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED
STATES

BUSINESS REPLY MAIL

First Class Permit No. 73 Peterborough NH 03458

POSTAGE WILL BE PAID BY ADDRESSEE

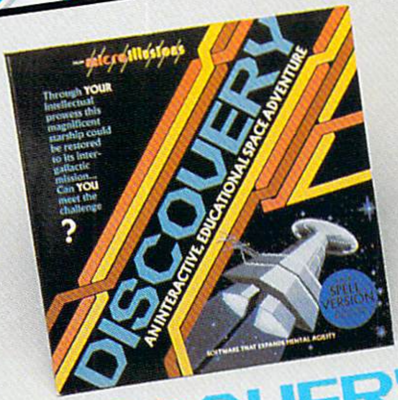
CW Communications/Peterborough
AmigaWorld
PO Box 868
Farmingdale, NY 11737

LAND OF LEGENDS

Mystery and excitement awaits your every move as you sojourn through one adventure after another in Land of Legends. Truly the most thrilling of all Dungeon & Dragon type adventures. Watch for this new release!

The FaeryTale Adventure™

Participate in the adventures of three brothers as they travel through a faery land full of vicious monsters, enchanted princesses, fearsome dragons and even a kindly wood cutter or two. Faery Tale Adventure will provide you with weeks of challenging exploration to conquer, so pick up your copy and begin today!



DISCOVERY

Learning becomes an exciting fun filled space adventure with Discovery! This entertaining learning experience is now available in Math, Spelling and the all new Trivia game version. So good, you will want to try all three!

OTHER PRODUCTS FROM MICROILLUSIONS

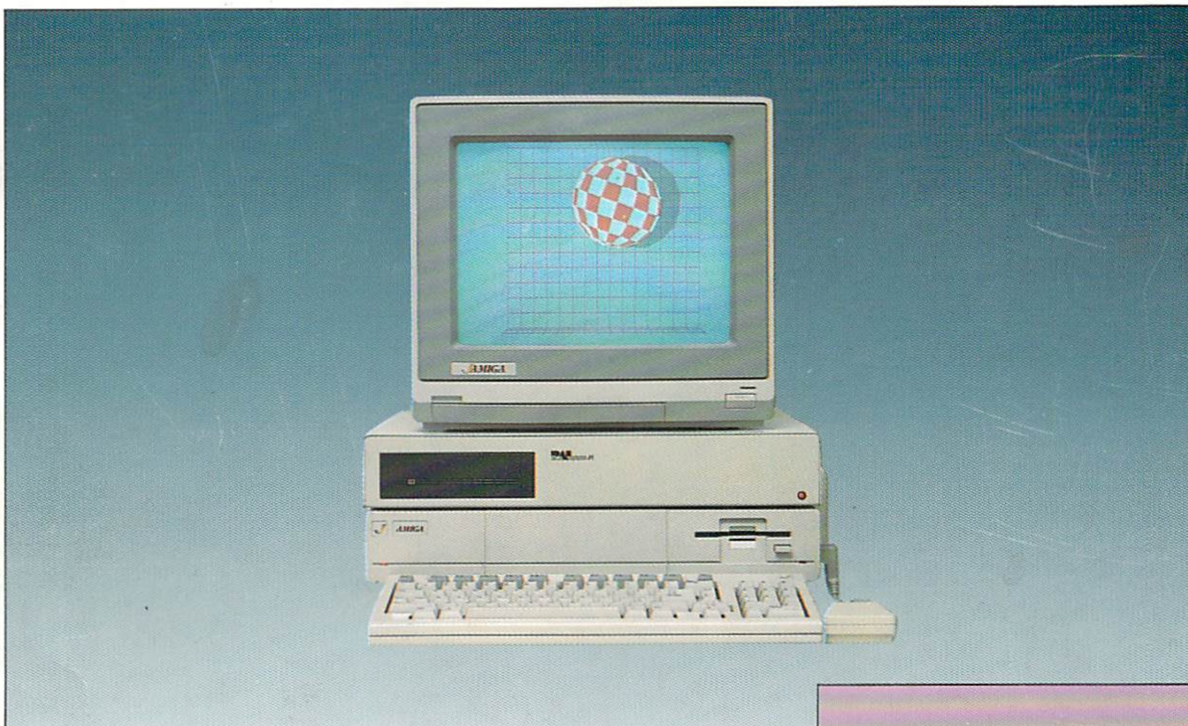
- **DYNAMIC CAD 2.3™**
  Now available
- **DYNAMIC WORD™**
  Watch for in May
- **MUSIC X™**
  Watch for in June
- **THE PLANETARIUM™**
  Watch for in June
- **MICRO MIDI™**
  Watch for in Fall
- **DYNAMIC PUBLISHER™**
  Watch for in the Fall

All of these products are now being developed for the Amiga, and will soon be available in other formats (C-64/128, IBM/PC, Apple, Atari ST).

microillusions™

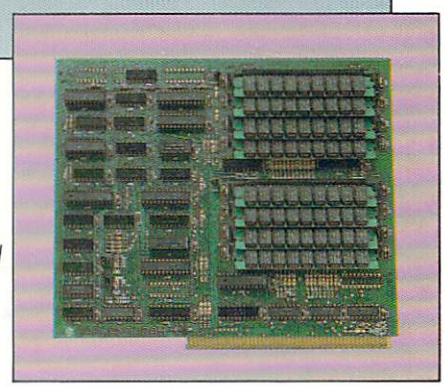
17408 Chatsworth St., Granada Hills, CA 91344, inside CA 818/360-3715 • outside CA 800/522-2041 • FAX 818/360-1464

There's a slim difference between the ordinary and the extraordinary. . . PAL JR.



- *1 MByte fast expansion RAM at C00000*
- *No wait states*
- *20 MByte hard disk drive*
- *True DMA controller with SCSI option*
- *Battery backed clock/calendar*
- *Open Zorro expansion slot*
- *Low profile, Amiga-beige case*
- *User expandable to 9 MBytes RAM*
- *Entire system auto-configures*
- *Suggested retail price $1,495.00*

2 MByte RAM CARD



- *High speed memory board*
- *No wait states during access or refresh*
- *Incorporates SIMM technology*
- *User upgradable to 8 MBytes*

BYTE by BYTE™

Arboretum Plaza II 9442 Capitol of Texas Highway North Suite 150 Austin, TX 78759 (512) 343-4357

Circle 300 on Reader Service card.